A Generic High-Level Specification Language for Non-functional Properties of Component-Based Systems

Alreshidi, Abdulrahman Nassar

Awarding institution:
King's College London

A Generic High-Level Specification Language for Non-functional Properties of Component-Based Systems

by

Abdulrahman Alreshidi

October, 2015

A thesis submitted in partial fulfilment for the
degree of Doctor of Philosophy

Department of Informatics
Kings College London, University of London

Acknowledgements

I would like to express my deepest appreciation to my supervisor, Dr Steffen Zschaler. Without his guidance and constant help this dissertation would not have been possible. I greatly appreciate both the time to invested and the ideas that contributed to my PhD, made the experience a truly productive one. His suggestions, detailed comments, and patient review allowed me to complete this thesis. I also express my warmest gratitude to my second supervisor, Prof Peter McBurney, for his endless source of ideas and motivating discussions.

I would like to express my gratitude to my parents for their absolute support over the years, which greatly contributed to my work and all their patience throughout this period. I also like to thank my brothers and sisters for their good wishes.

Finally, my big thank goes to my wife Maryam. I was embraced with infinite support in all aspects of my life and her patience during several time-intensive phases in the course of preparing this thesis.

Abstract

The component-based software development is helpful in providing reuse of the components and reducing complexity of software systems. Different components work together to produce a complete system that needs a good understanding of the way the components interact with each other. The components' reuse requires a high level specification, among other things for non-functional properties (NFPs) as these properties control the way these components co-ordinate with each other. The complexity of modern software systems demands a generic and flexible language for formal specification of the functional and NFPs of the system so that the different components in a system can have a well-defined behaviour expectation. The non-functional properties of component-based system are important part of specification because they highlight the non-functional perspective of the system. They also help in implementation of functional elements with constraints on the NFPs in consideration. The absence of specification of NFPs can render the system not usable because the functional implementation may not have considered the constraints for working environment of the system. This is because the component developer will have no clearly defined non-functional objectives of the system. The formal specification of NFPs for components and their interaction with each other can help implement reliable systems. Incorporating these design concepts in the language specification would describe the usage context of language features in clear and precise manner.

In this thesis, we developed a novel generic specification language (QML/CS) for NFPs of component-based systems. Defining such a high level specification language using a standard meta-modelling approach is challenging because its definition requires multi levels modelling. We employed deep meta-modelling technique to address this complex problem. We begin by discussing the key concepts used, then show how our meta-model is defined. In addition, we show how our meta-model for QML/CS overcame the issues of the standard meta-modelling language like UML and the mapping of a measurement to a concrete application. Finally, we show a prototype for QML/CS and discuss how the mapping of QML/CS expressions into TLA+ specifications can define the QML/CS semantics.

keywords: Non-functional properties; Model-Driven Engineering; Weaving Models; Multilevel modelling; Domain-Specific Languages; Component-Based Systems.

Contents

Abbreviations **xi**

1 Introduction **1**
1.1 Background and Motivation . . . 1
1.2 Problem . . . 3
1.3 Research Aim and Objectives . . . 4
1.4 Envisioned Solution . . . 4
1.5 Research Questions and Hypothesis . . . 5
1.6 Contributions of the Thesis . . . 5
1.7 Research Methods . . . 6
1.8 Overall Thesis Structure . . . 7

2 State-of-the-Art and Related Work **10**
2.1 Component-based Software Engineering . . . 10
2.1.1 Introduction . . . 10
2.1.2 Software Components . . . 11
2.1.3 Software Architecture . . . 14
2.1.4 Component-Based Development Process . . . 16
2.2 Non-Functional Properties . . . 19
2.3 Software Language Engineering . . . 21
2.3.1 Introduction . . . 21
2.3.2 Model . . . 22
2.3.3 Meta-Model . . . 23
2.3.4 Model Driven Engineering (MDE) . . . 23
2.3.5 Model Driven Architecture (MDA) . . . 25
2.3.6 Model Transformation . . . 27
2.4 Related Work . . . 27
2.4.1 QML . . . 28
2.4.2 NoFun . . . 29
2.4.3 CQML . . . 30
2.4.4 HQML . . . 30
2.4.5 CQML+ . . . 31
2.4.6 SLAngs . . . 32
2.4.7 UML SPT and MARTE . . . 33
2.4.8 CB-SPE . . . 34
2.4.9 Robocop . . . 35
2.4.10 Zschaler's Framework . . . 35
2.4.11 TADL . . . 38

2.4.12 E-Motion Observers . . . 38
2.4.13 Palladio . . . 39
2.4.14 ProCom . . . 40
2.4.15 The Framework of Jezek and Brada . . . 41
2.4.16 Descartes . . . 42
2.4.17 The Framework of Banerjee and Sarkar . . . 42
2.4.18 Discussion . . . 43
2.5 Summary . . . 44

3 A Meta-Model for QML/CS 46
3.1 Introduction to QML/CS . . . 46
3.2 Language Architecture . . . 47
3.3 QML/CS Meta-Model . . . 50
3.3.1 Context Models and Application Models . . . 51
3.3.2 Measurement . . . 54
3.3.3 Component and Service . . . 56
3.3.4 Resource . . . 59
3.3.5 Container . . . 63
3.3.6 System . . . 64
3.3.7 Extension of the OCL Meta-Model: QML/CS . . . 69
3.4 Summary . . . 70

4 Context Model Definition Ambiguity 72
4.1 Meta-Modelling . . . 72
4.2 Ambiguity of Context Model Definition . . . 73
4.3 Deep Meta-Modelling . . . 75
4.4 QML/CS Definition with MetaDepth . . . 80
4.5 Summary . . . 84

5 Specifying Mappings between Context and Application Models 86
5.1 Introduction . . . 86
5.2 Applying a Measurement to a Real Application . . . 87
5.3 Weaving Model . . . 88
5.4 Validation . . . 90
5.5 Summary . . . 94

6 Implementation 96
6.1 Technologies . . . 97
6.1.1 Eclipse . . . 97
6.1.2 EMF . . . 97
6.1.3 Xtext and Xtend . . . 98

6.2 QML/CS Component Architecture 99
6.3 Implementing Deep Meta-modelling 100
6.4 Implementing Mapping Model 102
6.4.1 Mapping Model Infrastructure 104
6.4.2 Validation Implementation 104
6.5 Integrating OCL into QML/CS Grammar 109
6.6 Summary 112

7 A Semantic for QML/CS 113
7.1 Translational Semantic 114
7.2 Translational Semantic Challenges 115
7.3 Translational Semantic via Epsilon 116
7.4 EGL Template of Translational Semantic 117
7.5 Code Generator Testing 120
7.6 Summary 122

8 Evaluation 123
8.1 A Case Study 123
8.1.1 Overview of the Case Study 124
8.2 QML/CS Specification 125
8.2.1 Measurement Designer's Perspective 125
8.2.2 Application Designer's Perspective 129
8.2.3 Platform Designer's Perspective 137
8.2.4 System Designer's Perspective 141
8.3 TLA+ Specification Generated 142
8.4 Limitations of Evaluation 144
8.4.1 Empirical Studies 144
8.4.2 Comparing to other Languages 146
8.4.3 Limited Scope of the Case Study 147
8.5 Summary 147

9 Conclusion 148
9.1 Addressing the Research Questions 149
9.2 Contribution to the Body of Knowledge 150
9.3 Future Recommendations and Lessons Learned 151

Appendix A :Specification of QML/CS and TLA+ 164
A.1 Specifications for Case Study 164
A.1.1 Context Models for Delta Time and Data Rate 164
A.1.2 Application models 166
A.1.3 Application Specification 166

A.2 TLA+ Specification Generated 184
A.2.1 TLA+ Specification of Application Interface 185
A.2.2 TLA+ Specification of Application 189
A.2.3 TLA+ Specification of DB Scheduler 201
A.2.4 TLA+ Specification of Web Audio Container 202
A.2.5 TLA+ Specification of Web Audio System 203
A.3 Translational Semantics Template of TLA+ specification 206
A.3.1 Context Model Template 206
A.3.2 Measurement Template 207
A.4 QML/CS Prototype Screenshots and Code sample 209

Appendix B :Grammar of QML/CS language in Xtext and Multi-level Modelling in MetaDepth 211
B.1 Complete Grammar of QML/CS language in Xtext 211
B.1.1 Grammar for QML/CS 211
B.1.2 QML/CS grammar 212
B.1.3 Extended OCL grammar 217
B.2 Complete Example Specifications of a Meta-model for QML/CS . . . 224

List of Figures

1.1 Research Design in this Study 7

2.1 The Component Forms derived from [25] 13
2.2 The Dependencies in Component Architectures derived from [25] . . . 15
2.3 Component-based Development Process derived from [25] 17
2.4 Development process for NFPs overview derived from [89]. 18
2.5 QML/CS Roles. 19
2.6 The four levels meta-modelling architecture of MDA as defined by OMG, derived from [77] 24
2.7 M0 Instance derived from [100]. 26
2.8 M1: Model derived from [100]. 26
2.9 M2: Model of model derived from [100]. 26
2.10 M3: Model of model of model derived from [100]. 27
2.11 the specifications derived from [110]. 36
2.12 System model derived as a UML diagram [110]. 37

3.1 The high-level organisation of the QML/CS meta-model. 48
3.2 Context and Application Meta-Model 52
3.3 Example of Context Model. 53
3.4 Example of Application Model. 54
3.5 Measurement Meta-Model. 55
3.6 Component/Service Meta-Model. 58
3.7 Resource Meta-Model. 61
3.8 Container Meta-Model. 64
3.9 System Meta-Model. 67
3.10 OCL meta-model derived from [83] 70
3.11 Extended OCLCallExpression 70

4.1 Example of Library Meta-modelling Hierarchy. 73
4.2 The UML two levels representation of the *Class*, *serviceOperation*, *Operation* and *getData*. 74
4.3 The UML two levels representation of the *ServiceOperation* and *getData*. 75
4.4 Ontological and Linguistic Classification 76
4.5 Deep Instantiation 78
4.6 The *Clabject* representation of the of Library Item. 79
4.7 The *Clabject* representation of the *serviceOperation* and *getData* derived from [46]. 80
4.8 Modelling *ServiceOperation* and *getData* using Deep Meta-modelling. 81
4.9 Deep Meta-modelling derived from [30] 82

5.1 An Example of the Mapping between Application and Context Models 88
5.2 Mapping Meta-Model. 91
5.3 Example: Mapping Model between context and application models. . 92

6.1 QML/CS Component Diagram 100
6.2 Implementing *Clabject* concept in QML/CS derived from [111]. 101
6.3 *getData* and *Operation* Lifting. 102

7.1 An Overview of a Model to Text Transformation via EGL 117

8.1 Web audio store architecture derived from [11] 124
8.2 Context Model for Response Time (RT). 127
8.3 Context Model for Execution Time (ET). 127
8.4 Application Model for Audio Rental Service. 131
8.5 Application Model for Web Form Component. 132
8.6 *RAOp2RtMappingModel* for Validating *RentalAudio* Operation. . . . 135
8.7 Resource Model for Database resource. 139

A.1 Context Model for Delta Time (InRT). 165
A.2 Context Model for Date Rate (DR). 165
A.3 Application Model for Audio Store Component. 167
A.4 Application Model for DB Adapter Component. 168
A.5 Application Model for User Management Component. 169
A.6 Application Model for Ogg Encoder Component. 170
A.7 Application Model for Encoding Adapter Component. 170
A.8 Application Model for MySql Client Component. 171
A.9 *UFOp2ETMappingModel* for Validating *uploadFile* Operation. . . . 172
A.10 *SOp2EtMapping* for Validating *subscribe* Operation. 173
A.11 *ROp2EtMapping* for Validating *read* Operation. 175
A.12 *AOp2EtMapping* for Validating *authenticateUser* Operation. 177
A.13 *EAOp2EtMapping* for Validating *encodeAudioData* Operation. . . . 179
A.14 *PEOp2EtMapping* for Validating *ProcessEncoding* Operation. 181
A.15 *SAOp2EtMapping* for Validating *storeAudioFile* Operation. 183
A.16 QML/CS Prototype Screenshot 1 210
A.17 QML/CS Prototype Screenshot 2 210

List of Tables

2.1 Comparison of QoS Specification languages and Frameworks 45

Abbreviations

ADL	Architecture Description Language
SM	State Machine
CASE	Computer-Aided Software Engineering
CBSE	Component-Based Software Engineering
CB-SPE	Component-Based Software Performance Engineering
COMQUAD	Components with Quantitative properties and Adaptation
CORBA	Common Object Request Broker (ORB) Architecture
CQML	Component Quality Modelling Language
CQML+	Extended Component Quality Modelling Language
CPU	central processing unit
Eclipse	Eclipse IDE and RCP
Ecore	meta-meta-model included in EMF
EJB	Enterprise JavaBeans
EMF	Eclipse Modeling Framework
IDL	Interface Definition Language
ISO	the International Standardisation Organisation
MARTE	Modeling and Analysis of Real-Time Embedded Systems
MDA	Model-Driven Architecture
MDE	Model-driven engineering
OCL	the Object Constraint Language
OMG	Object Management Group
pUML	precise UML to provide a precise and formal basis for the UML
PCM	Palladio Component Model
QML	Quality Modelling Language, a specification language for QoS.
QoS	Quality of Service
SLA	Service Level Agreement
SLAng	a language for specifying SLAs
TLA	Temporal Logic of Actions
TLA+	Extended Temporal Logic of Actions
UML	Unified Modelling Language
WCET	worst-case execution time
XMI	XML Metadata Interchange
Xtend	a programming language on top of Xtext
Xtext	framework for development of programming languages and DSLs

1 Introduction

The success of a software system, relies not only on functional aspects, but also on non-functional properties (NFPs) which are important for a whole development. Therefore, it is desirable to predict and analyse NFPs in the first phase of the development as this would save the developers of the system from re-doing the design and implementation again as well as any cost in the final phase of development. This chapter introduces the background and motivation of the current research by showing the key research problem as well as the significant contributions the current research aims to make.

1.1 Background and Motivation

Maintaining modern software system becomes more difficult in general when the complexity of system increases; particularly it becomes even more challenging when time of building a software and its distribution in a competition software market is reduced. Component-based Soft Engineering (CBSE) [100] is a software development approach that plays an important role in producing reliable complex software systems efficiently. The idea of creating component-based systems belongs to reducing the complication of the large systems by decomposing them into smaller components so that the requirement can be managed properly. Also it helps in achieving performance goals with getting these components to work together. Therefore it is inherent that component-based systems are complex in nature and their requirements are heavy-duty in nature because it needs not only to mention that requirements of a component but also its interaction with other components in the system with expected in and out behaviour of the component. The CBSE approach focuses on providing guidelines to handle these complications at different stages of development of a component-based system. It is based on the concept that different plug-and-play software components work together to complete functionality of the software system; these software components work together with each other by using the exported interfaces from each component. Thus, this integration mechanism helps in creating

many complex, yet efficient systems by combining different components in order to build a fully functional software system.

The quality attributes of the overall system depend on the quality service attributes of software components. The specification of quality of service (QoS) attributes is equally important for a sustainable component-based system [86]. The specification of QoS will help implement these requirements effectively so that a feature is implemented and then validated against all constraints attached with the requirement. This will add to the test coverage of the system and known system performance in different deployment environments. It also results in a predictable system behaviour based on the implementation done for QoS attributes that is guided by the formal specification of those attributes.

Modelling the QoS for component-based systems has been an important research focus for some time, but two main aspects are as follows: first, the absence of a standardized method to specify NFPs, as a result different people describe the NFPs differently and which may also result in misunderstanding on the part of the component developer. Second, the low level of abstraction of some specification languages makes them practically unusable as they cannot be used to specify NFPs fully. The NFPs of component-based system are important part of specification of a software system because they define the behaviour of the system in different working environments. So, incomplete specification of NFPs may reduce the acceptance of a software system as the behaviour of this system in production environment will be uncertain.

The quality assurance team may have a different perspective than the development team when reading functional specification, if a consistent NFPs specification is not available. Although component developer will test the functional implementation but this process may not have complete test coverage and therefore implementation is not well tested. Therefore, the placement of such a system will be at a high risk of failing because it has not been tested enough to comply with different working situations and resource requirements. A standardised pattern of NFPs specification will help in developing a consistent specification as compared to different styles or formats used without following a standard process. The standardised specification will reduce the human error because everyone will use the same standard to specify the NFPs and will provide a uniform access to the information related to non-functional aspects of the system. This will also increase the chances that the component developer as well as the quality team will have better chances in building a stable and scalable system owing to the fact that the non-functional attributes will be implemented and tested.

Modelling NFPs of component-based system increases the success chances of component-based system. The quality modelling language for component-based systems (QM-

L/CS) is a specification language for NFPS of the CBSEs sketched by Zschaler [109]. We take motivation from the idea presented in [109] and proposed a fully developed language specification, which includes the formal specification of the language and an improved definition of some of its concepts (e.g., model mappings, a refined specification of measurements). The significant motivation driving the development of QML/CS was the existing need for a ready-to-use, comprehensive, versatile and highly expressive modelling language which is suitable for the development of modelling NFPs of component-based systems. To qualify this more precisely, QML/CS was intended to be a language that:(1) is built on key concepts and specifications presented in Zschaler's framework [109],(2) integrates best practices for OCL metamodel, (3) is well specified and documented specification language.

1.2 Problem

Several studies have contributed to elaborate on the extensive effort of modelling NFPs of component-based systems and how it can implemented [11, 13, 20, 32, 42, 58, 94]; these studies often focus on developing specialised methods to measure a specific non-functional property of component-based systems, such as performance or reliability. Although there are some formal specification languages and approaches [3,39,40,48,64,78,87,95], their semantics are not formally defined. In addition, there are frameworks designed for the specification of NFPs [9, 53, 103, 110] that try to provide formal specification to generically specify NFPs of component-based system.

However, their low level of abstraction makes them practically unusable. The low level of abstraction or formalisation is related to the length and complication of specifications for rather simple language features because such languages like TLA+ do not have structures to represent concepts at high level. It results in very detailed specification of even simple concepts. The extent of specifications require the user to have all that knowledge even when it is not needed based on the requirement of specifications. That discourages users from using such a language because they have to learn a lot about the language itself rather than focusing primarily on the actual objective of writing specifications. That is why languages like TLA+ are not practically usable. Therefore, a generic and usable quality modelling language with formal semantics that can specify NFPs of component-based system will help in improving the techniques to address quality of service attributes.

There are different users who should be using the language to specify the NFPs and each user has a specific role in the software development life cycle. Two key types of users interacting with a specification language are application designers and compo-

nent developers. Component developers can specify NFPs of components they have developed using this generic language. Application designers also have the ability to apply measurement in order to offer guaranteed NFPs to their own concrete applications.

In his research, Zschaler [110] identified key concepts and the type of specifications that could be used to formally specify NFPs of component based systems. The present work extends these concepts from previous work to design a language that is more readable, can hide the complexities in abstraction and allows formal specification of NFPs practically. This is accomplished by replacing the low level formalisation in Zschaler's TLA based framework [110] with a high level specification language.

1.3 Research Aim and Objectives

The aim of the current thesis is to define a generic usable language based on Zschaler's framework. The present work inspired by the previous research [109] aimed to formalise the specification language for QML/CS so that it can be practically usable to model NFPs of component-based systems generically. The specific objectives of this study are:

- To critically review the existing specification languages and tools which models NFPs of CBSE.
- To identify a suitable meta-modelling approach of defining QML/CS specification and providing its formal definition.
- To provide a tool in order to demonstrate the usability of quality modelling language of component-based systems.

Our novel contribution will focus on defining a well-specified language as well as providing a practical specification in modelling NFPs of component-based systems.

1.4 Envisioned Solution

This thesis defines a practical and generic language for the specification of NFPs of component-based applications. This language will help different users like application designers and component developers to share same standard for specification of NFPs. The crucial steps towards building a quality modelling language for NFPs of a component based system in thesis involves utilizing and extending core concepts introduced in [110]. It also includes use of meta-modelling and the definition of a

domain-specific modelling language for formal specification of NFPs. Component-Based Software Engineering (CBSE) is very focused on specification of components and applications so that a specification-heavy approach would be a good fit. We are aiming for the specification-based approach in line with CBSE approach, which is for specification of components where set of components can be specified separately and then tagged together to complete system specification.

1.5 Research Questions and Hypothesis

The key questions driving the work are:

1. Is it possible to specify the QML/CS specification language using a meta-model, if so how?
2. Whether a usable QML/CS specification language can be defined?

To be able to achieve these objectives, it is important that hypothesis specifically reflects the expected requirement so that research questions can help in evaluating if the objective was practically achieved. The overarching hypothesis for this study is as follows:

- Meta-modelling approach can specify QML/CS, which is important to define quality of service of the software system.
- The QML/CS based on meta-modelling approach that can formally specify NFPs is practically usable.

1.6 Contributions of the Thesis

In this section we point out the contributions of this thesis to the research on formal specification of NFPs of component-based systems. The thesis makes the following contributions to the research:

- Major Contributions:
  - *The thesis provides a novel specification language for the formal specification of NFPs of component-based systems (Language Definition for QML/CS)*[1]
  - *Applying deep meta-modelling to define QML/CS.*

[1]The implementation link of our tool: http://a-alreshidi.github.io/QML-CS/.

  - *The ability to capture and validate simulations between state machines in a mapping model.*

- Secondary Contributions:
  - *The thesis defines the Semantics Translation to TLA+.*
  - *A working prototype as a basis for the integration of QML/CS in future language specification tools.*
  - *Integrating OCL into QML/CS Grammar.*

1.7 Research Methods

We use a constructive software engineering research approach, which uses artefacts as a proof of concept that a specific question can be answered and show how it should be answered. The focus of this thesis is the development of modelling NFPs for component-based systems. To this end, we carried out the following phases, as can be seen in Fig 1.1:

- **Phase 1** (Critical Review: State-of-the-art):
  - The basic research approach in this thesis is to study various existing quality languages and frameworks like CB-SPE, Palladio, Descarties, QML, CQML and CQML+ [3, 11, 13, 40, 58, 87] and evaluate them based on specific criteria such as formal semantics, genericity, practicality, application domain and complexity.
  - Identifying problems: Based on initial state-of-the-art investigations and exploration of low level of abstraction; it is realised that QML, CQML, CQML+ and the Zschaler's framework are not usable by the developers.
- **Phase 2** (Meta-Modelling Approach): We take an inspiration from examples presented in [110]for a specification language QML/CS along with introducing some meta-types based on Zschaler's approach that identifies key concepts and type of specification, and derive a meta-model for QML/CS. We explore various techniques of developing a meta-model for QML/CS.
- **Phase 3** (Meta-Modelling Challenges): In order to successfully model such a requirement where the entities can be specified at multiple levels and their existence depends on the relationship they have with their entities, we need a modelling technique that can represent more than one existence of same entity based on the role of that entity in that specific context. An existing modelling

technique called *Clabject* comes handy in modelling such entities and give discrete representation to their both roles of a class and an instance. Weaving models [6] can be used to define a relationship between a source model and a target model with certain mapping conditions based on predefined rules which can be user-defined. Weaving model contains a set of links between elements of a model and elements of another model [33].

- **Phase 4** (Prototyping Approach): The proposed solutions in phase 3 requires that a tool support for the language is implemented, which can be used to specify NFPs and it should support mapping and modelling solutions proposed in this thesis. The prototypical implementation requires to deal with shortcomings of the current tooling infrastructure.

- **Phase 5** (Testing and Evolution): To evaluate and test the ability of defining NFPs of component-based system via QML/CS language, we selected an industrial application, called Web Audio Store [11]. This case study is specified via QML/CS language.

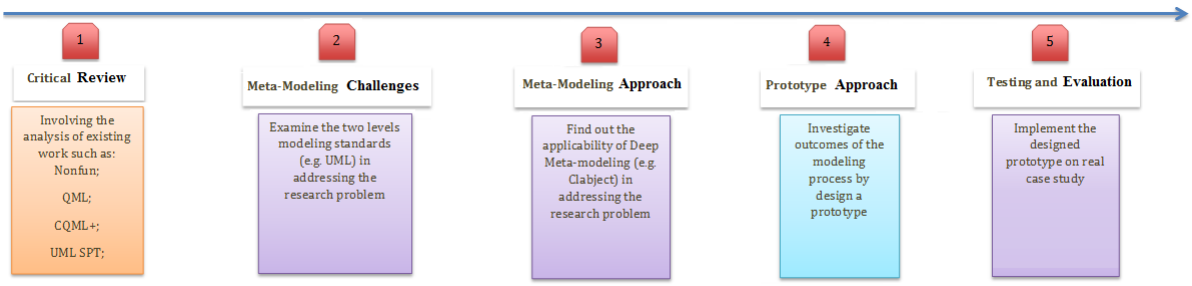

Figure 1.1: Research Design in this Study

1.8 Overall Thesis Structure

This thesis is designed and structured to comprise nine discrete but consecutive chapters. A brief summary of the content of these chapters is described as follows:

Chapter 2 Literature Review and Related Work: In literature review section, we review a background and previous works related to the research domain. In related work section, we discuss the issues including but not limited to formality of quality of service of modelling languages, practically of modelling languages and genericity, which is the ability to allow NFPs for component-based systems to be defined as required.

Chapter 3 A Meta-Model for QML/CS: This chapter gives an overview of the meta-model presented in this thesis. It defines the quality modelling language, a specification language for modelling NFPs of component-based software (QML/CS) through the creation of a new meta-model.

Chapter 4 Applying Multi-Level Modelling: This chapter presents the first key challenge we faced during the development of QML/CS language. The limitation of defining QML/CS meta-model using UML standards is shown. It also provides solution of applying deep meta-modelling using the technique*(Clabject)*. The research question 1 is addressed in this chapter.

Chapter 5 Specifying Mappings between Context and Application Models: This chapter shows the second problems we faced during the development of QML/CS language. The issue of validating the parameter of measurement with an appropriate type based on the measurement definition is presented. The chapter also provides solution of the presented problems using the technique weaving models *(Model mapping)* methods to specify mappings between context and application models. The research question 1 also is addressed in this chapter.

Chapter 6 Implementation: This chapter presents the implementation of the concepts presented in the chapters 4 and 5. It also shows the prototype for the Quality Modelling Language for Component-Based Systems (QML/CS), a lexical language for specifying NFPs. The prototype implementation for the quality modelling language (QML/CS) tool is described, the main parts of the prototype are outlined as well as the internals of the tool implementation are explained. The research question 2 is addressed in this chapter.

Chapter 7 A Semantic Translation for QML/CS: This chapter shows the semantic translation. The semantics are given for the specific form of the language chosen for implementation, which is QML/CS. The actual semantic translation uses Epsilon transformation language for QML/CS specifications. It defines the meaning of any QML/CS specification in terms of our QML/CS language and mapping a QML/CS specification consisting of characteristics and OCL expression into a TLA+ specification. This semantic translation is always parametrised by a context model, an application model and classes.

Chapter 8 Evaluation: This chapter presents the description how the prototype is evaluated. It also describes the result from the evaluation. It shows the findings from applying the refined factors and analytic hierarchy process in a case study in order to evaluate that the QML/CS specification language is practically usable to specify NFPs of component-based system. The research question 2 is also addressed in this chapter.

Chapter 9 Conclusion: This chapter summarises the entire thesis by providing the answers to the research questions and presenting the contribution to the body of knowledge. It also discusses the limitations of the research together with recommendations for future research.

2

State-of-the-Art and Related Work

In this chapter, we review existing literature related to the area of Component-Based Software Engineering (CBSE) and Quality of Service. Section 2.1 presents component-based systems, showing some of their component forms and related terms. Section 2.2 provides elaboration on the functional and non-functional properties (NFPs) and linking them with the scope of this thesis. Section 2.3 introduces the concept of model, meta-modelling and explain how it relates to Model Driven Engineering (MDE). Section 2.4 provides discussion of related work in the area of specification languages for specifying NFPs.

2.1 Component-based Software Engineering

In Subsection 2.1.1 we provide an introduction about CBSE and its importance. Subsection 2.1.2 shows the meaning of the concept software component, Subsection 2.1.3 explains software architectures. Then, in Subsection 2.1.4 the component-based development process is discussed.

2.1.1 Introduction

The complexity of software schemes has been increasing over the years. Such intricate systems have to manage a large number of tasks and it takes a considerable amount of time to manage errors identified. Thus, it is essential to counter this increased level of intricacy by introducing techniques to manage it. Component-based Software engineering (CBSE) [45] has been observed as a promising example of such a technique [23]. CBSE is a software engineering approach that is concerned with the development of software from reusable components and components development. The main aim of CBSE is to minimize the dependency between different software components. Zschaler discussed the two important views concerning the benefit of CBSE [110]. One is based on Szyperski's view [100] which is about ability to reuse parts of code written by third parties. The second is based on Cheesman and

Daniels [25] who see the benefit of CBSE as independent self-contained modules that have flexibility to work with each other and are not tightly coupled.

The CBSEs have their architecture designed on the basis of the development of independent and loosely coupled components of the system. The connection among different components of software system can be established using interfaces so that one component is able to offer services, which another component requires. The advantage of CBSE is that it reduces the complexity of large and complex systems and breaks them down into independent modules that work together to give the service offered by the full system [45]. With the increased demand of ready-to-use components that can be plugged into an existing system to provide the functionality offered by the component, CBSE plays an important role and is very successful in linking those independent components into one fully working system. These components are integrated by application contractors to create complete solutions. In a software component industry, if there is a negotiation taking place about components, this requires specification of their high-level properties. Component developers must present a description to create greater understanding of the background in which their components will be deployed [15]. Conversely, it should be clear to application designers how they can create specifications for distinct components to clearly understand the properties of the system.

2.1.2 Software Components

In this subsection, the concept of software component is reviewed and some discussion on fundamental definitions of a component provided based on literature.

Meyer [69] stated that software components stick to the principles of software objects due to the evolution of object-oriented technology. The following are the three key principles of software objects. The first one is a state, which represents the data stored in software objects. The second principle is a behaviour, which is about accessibility of a function to use a software object and manipulate its state. The third principle is an identity, and that is having a unique identity with no consideration of its inner state.

The three principles outlined above are extended by software components and the focus moves from the implementation of software components to their specification [25]. Interaction with components is done through their interfaces and there are two kinds of interfaces, provided and required interfaces. Components can be accessed by users using provided interfaces. One component can connect to other components using its required interfaces. The distinction between the specification of a software component

by its interfaces and its implementation by code is obvious. A component implementation can be easily replaced by another one, provided the declared interfaces of the replacing component adhere to same component specification.

Evidently, a component is simply a piece of software. The most cited and important definition of a software component is Szyperski's definition [100], which is

> "A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties."

Cheesman and Daniels [25] have observed that providing the definition of a component could be complex because it will need specification of how other components can work with it. Therefore, it would be more clear to understand if we examine the different forms which a component could take. It is also important to define the forms clearly and specify relationship between those forms so that the transition from one form to the other can be understood. Few of the criteria that should be considered for transition between the forms include the component standard; i.e., a component should obey the rules established as an environmental standard. In other words, the component will be used if it meets certain standards; e.g. Enterprise Java Beans (EJB) [71], Common Object Request Broker Architecture CORBA/ZIOP [41] and Microsoft's COM+ [70]. Large businesses tend to use their own components, which have been defined for them; therefore, components need to be measured by a determined standard.

In addition the specification and utility of a component also identifies if a component is compatible with the form of another component. For example, with regard to the fuse used in a power supply for a present day computer, a 5-amp fuse would not be suitable if the requirement was for a 15 amp fuse. Both would fit without difficulty, however, the 5-amp fuse would be damaged, as it has been designed to meet a different capacity. It means that all the interfaces a plug has to use from the fuse to provide the function should match the required standard otherwise even one missing or differently specified interface can render it not working. It indicates that specification of components, which provides clear specification of required and provided interfaces, should consider the specification standards of the components it may work with in the future. The clear and precise specification of components and its required and provided interfaces is the only way to develop flexible independent components.

Cheesman and Daniels [25] classified four component forms as shown in Fig.2.1.

1. The first form is a component specification, which gives the behavioural description of objects, along with the specification of implementation and deployment.

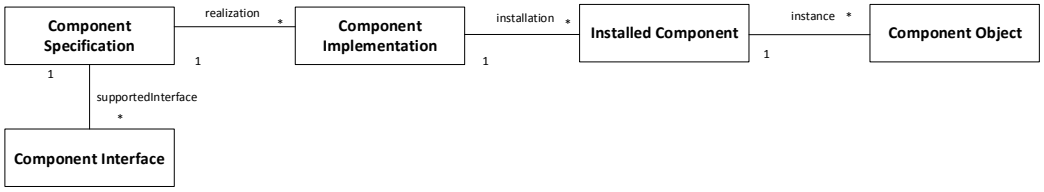

Figure 2.1: The Component Forms derived from [25]

Component interfaces are parts of component specifications. Component interfaces can declare the services provided by the component to users. In addition, interfaces can also declare the services required by the component from other components.

2. A component implementation is the second form of a component that deals with the realization of the first form of a component by a contract. This realization contract provides a way of the negotiation between the first and second forms of the component. A single component specification can have a number of implementations, provided that they show the same publicly defined interfaces.

3. The third form of a component is an installed component, and that is a copy of the component implementation. There can be a number of installed components for one component implementation. An Installed component can be assembled with other components to instantiate it on runtime environment.

4. The last and forth form is a component object, which is an instance of an installed component to be used. An installed component can have a number of component objects, and these component objects can only be differentiated through their unique runtime state. To understand the concept of many instances for the same installed component, we can consider example of a "Windows Explorer" component that is installed on Windows computers to be able to access files on the system. The programmers would have implemented it based on a textual specification provided to them and then used compiler to create an executable file. Because there are could be several folders in a file system, the user may be interested to see contents of two different folders at the same time. The user will start two independent instances of "Windows Explorer" component with each instance pointing to a different folder. It indicates that it is not the installed component itself rather its instance that is used to consume the functionality provided by the component.

Zschaler [109] divided NFPs into two types; intrinsic and extrinsic properties. The intrinsic properties apply to component implementations and depends on how the

implementation is done and what kind of resources it uses; one such example is execution time. The extrinsic NFPs give a service-level perspective and are attached more with the user's expectation than the implementation details of the service. Response time is an example of extrinsic non-functional property that concerns only about the time it takes for a service to complete regardless of how it is done and the way its architecture may have been designed. The NFPs like execution time and response time can be attached with a component form based on the type of property and the form of component it belongs to. Execution time is attached with component object and not just the installed component because it can only be determined when a method is called and the time to execute it can be calculated. Other NFPs such as resource consumption can be linked with installed component form because it can check at installation time if the system has the required resources available.

2.1.3 Software Architecture

The notion of software architecture is defined in [66] as:

> "The fundamental organisation of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution".

This thesis helps improving the specification of NFPs of the components so that the architecture of the CBSEs is more clearly defined. This is achieved by formal specification of the NFPs in a consistent way; this in turn will facilitate the interface specification for both required and provided interfaces of a component to effectively work with other software components in the system. A clear definition of a software architecture concept is also given in [25]. Cheesman and Daniels distinguish a system architecture from a component architecture as follows.

- The system architecture: The system architecture is the structure of elements that form with each other a complete software system, which contains the responsibilities of these elements, their interconnections, and possibly the appropriate technology. The system architecture may be made up of various architectural layers. For example, the user interface of *Java EE* is performed using *Java Server Pages*, the state of user interaction that is held by the user dialogues is performed with *Java Beans*, the business logic that is represented by the system services is implemented via *J2EE session beans* and the persistency that is ensured by business services is implemented with *J2EE entity beans*.
- The component architecture: The component architecture is included in the system architecture. It is defined as a set of application-level components,

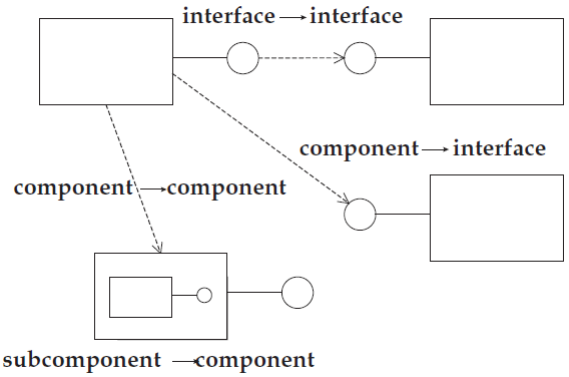

Figure 2.2: The Dependencies in Component Architectures derived from [25]

their structural relationships and their behavioural dependencies. The meaning of structural relationships here is the associations and inheritance between component specifications and component interfaces, and composition relationships between components. The meaning of behavioural dependencies is the dependency relationships between components and other components, between components and interfaces, and between interfaces, as shown in Fig 2.2. For instance, a component architecture may refer in the application to the server part. Component architecture can be seen as a logical concept rather than technical realisation. The architecture contains a link between components that defines the contract between the components and helps the components to comply to each other when providing or consuming a service. These links are called connectors, which can be expressed via unified modelling language (UML) component diagrams.

Component Specification Architectures

A component architecture can have different perspectives like implementation perspective, specification perspective and the component object perspective with each perspective describing a specific usage situation [25]. The component specification architecture will be focused around definition of interfaces and the specifications required to allow the integration of the component with other components in the system. This specification will also indicate dependency of the component on other components and describe the way that dependency is resolved. A definition with similar understanding is provided by Cheesman and Daniels [25]. They state that the specification architectures has an important rule, which is:

> "Any dependency emanating from a component specification is part of the definition of that component specification and must be adhered to by all implementations."

Since many different implementations can be done for the same interface specification, the dependency of a component defines the pre-requisite for every implementation that is supposed to be complied to before an implementation is considered acceptable. It means the dependency of a component works as common point of interest and binds different implementations of the component. The principle of component substitution also depends on the component dependency and its correct implementation because a component can substitute another component only if it respects all the interface expectations as well as dependency considerations [25]. The component specification will contain all the information required to be met by different implementations of that component in order to allow its replaceability.

Contract Specification

A contract defines the protocol or steps that must be taken by the client of a component to be able to use it. It lists the steps as well as pre-requisites for using a service provided by the component. There are generally two type of contract specifications that exist for a component and both are important for the component to work properly and other components to be able to use the service provided by the component [25]. The first layer is the usage layer where the contract specifies the pre-requisites for any user to meet before they can use the interface. They need to request an instance of the interface and then know what parameters need to be passed and what output will be returned. So this usage contract specification will particularly address the information required by the client who wants to use the component. The second layer of the contract is the realisation layer where the interface definition is mapped to the implementation so that the service can be provided as expected. There can be different implementations for the same contract; thus it is important the realisation contract is clearly specified so that ambiguity of implementation can be avoided. This contract specification facilitates component substitution because all the components that can substitute other components must adhere to the contract specification so that the client uses the substituted component without having to change anything [25].

2.1.4 Component-Based Development Process

In software development processes, component-based software development process is different from object-oriented development process [100]. The task of developing software artefacts is separated into two roles. The first one is component developer who concerned with developing single components. The second one is the software designer who is concerned with assembling those components to build up an application. In [25], a component-based development process model is proposed based on

the Rational Unified Process (RUP) as can be shown in Fig 2.3. In this development process model, the specification and implementation of a component-based software system is described. This process is concerned with structuring a working system from requirements. In contrary, the model disregards the concurrent management process. It is related to time planning and controlling. In Fig 2.3 each box represents a workflow, a thick arrow between those boxes shows a change of activity and thin arrows denote flow of artefacts among the work-flows. The work-flows allow reverse steps; incremental or iterative development based on prototypes is also allowed. The main work-flows are requirements, specification, provisioning, assembly, test and deployment.

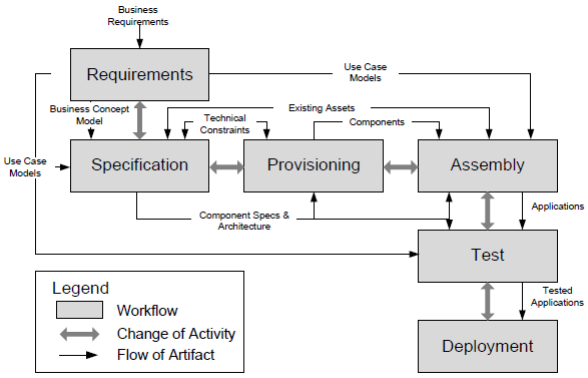

Figure 2.3: Component-based Development Process derived from [25]

Rottger and Zschaler [88, 90] propose a methodology that extends the view of application development in Model Driven Architecture (MDA) [81] to NFPs. Their overall software development process for NFPs is presented in Fig 2.4. Once the requirement analysis is complete and the requirement specification document is available for modelling, the Application Designer will use this as reference document to specify NFPs of the system and the constraints attached with the requirements. The application designer toggles between modelling as well as refining NFPs of the components and of the components' environment. The main concept of this approach is the division of measurement description usage [89]. Furthermore, in their approach the definition of measurement is separated from measurement usage, which leads to having two roles of measurement designer and application designer. As a result, measurement designer and application designer roles helps in combining NFPs specifications to the system specifications.

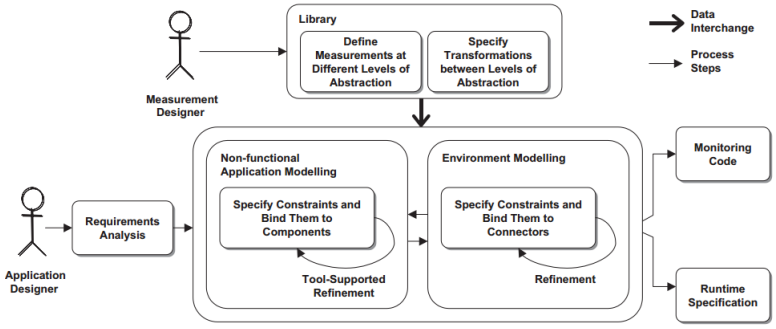

Figure 2.4: Development process for NFPs overview derived from [89].

We took inspiration from Figure 2.4 to describe the process of writing the specification using QML/CS presented in this thesis. We redesigned the roles and their work flow in using QML/CS to specify NFPs from NFPs discovery to formal specification. There are different aspects of specification like measurement, application, resource, container and system when it is done using QML/CS and work flows for each aspect is specified so that each aspect constitutes a role that can be given to the personnel working on it. These work flows are elaborated in Fig 2.5 that highlights different roles and how do they interact with QML/CS to specify NFPs. We can derive from this that structure of QML/CS allows different people to work together on a project specification independently while complementing each other.

The application designer will derive NFPs from the requirement specification and use this as a reference document to specify NFPs of the system and the constraints attached with the requirements. Since this QML/CS is designed for component based systems therefore the application designer will have to model the requirements as services and components and then attach their relevant NFPs like response time and execution time. The application designer also needs to access the measurement repository created by the measurement designer, who has identified all the measurements in the system and suggested constraints for each measurement based on requirement specification document. The application designer will connect the measurements specified by the Measurement Designer into the application components and services so that non-functional specification of the system is complete. The completely specified system document will then be made available for the Component Developer so that this document can be used as guideline to implement the components attached with these measurements and their constraints on the component and service operations. This is done by the application designer where the operation is passed to measurement as concrete argument, then constrains is placed over such a measurement. Using QML/CS language, the application designer specifies what is needed for a system based on the requirements of the customer. Flow for four key users of the

language, indicates how they initiate and complete formal specification (see Fig 2.5).

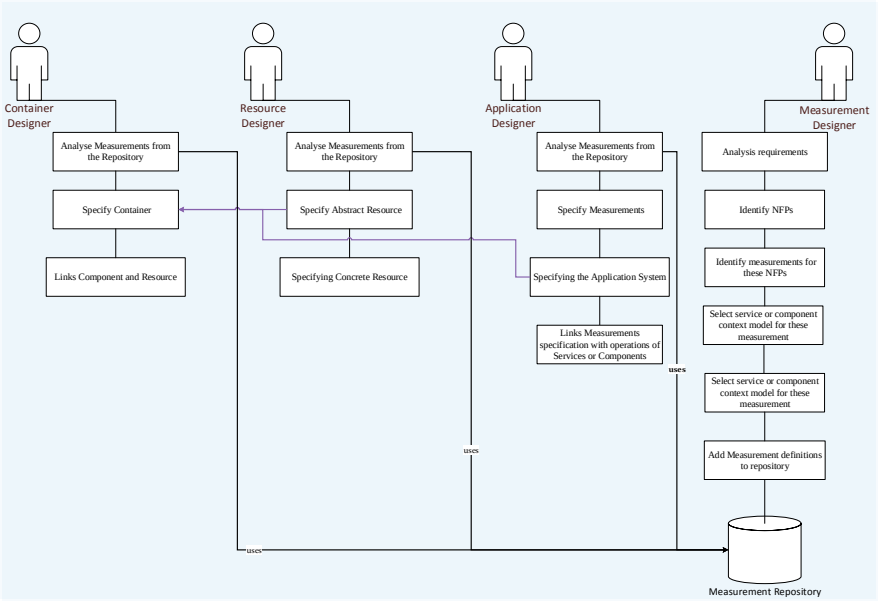

Figure 2.5: QML/CS Roles.

2.2 Non-Functional Properties

When a system is being developed, its specifications are prepared defining the functional and non-functional requirements of the system. The functional requirements outline the functions the system should implement whereas non-functional requirements are concerned with the behaviour expectations from the functional implementation [63]. The non-functional requirements are decomposed into one or more NFPs because the requirement is merely an expectation that should be clearly described in the form of NFPs. For example, a non-functional requirement from a banking system can be that "The system should not take more than 10 seconds to complete a deposit transaction"; this requirement can be fulfilled if these constraints are part of the specification so that component developer can consider it implementation. This is because they both will contribute to measuring such properties.

The NFPs are those properties of the system that define operation of the system on top of how it behaves and what functionalities it is supposed to offer. They are an important part of the system architecture because they define not only the constraints on the different components of the system but also define the standards and protocols

that should be followed when working together. While functional properties of the system provide information about functions or features of the system, the NFPs will represent the behaviour of the system in different operational environments and what difference of behaviour can be expected while operating in each environment [26].

Since the NFPs exhibit different perspectives of the system from quality of service point of view, it is important to be able to quantify them so that they can be measured. Different perspectives of the system will have different measures or units attached because some aspects are measured in time while others are measured in numbers; some even need nominal classification or categorisation of the property's quantification. Not all the NFPs can be measured as discrete values (for example, accuracy). Each system will have an impact on the way we quantify its NFPs because its functional elements will define what needs to be measured. Although each system will have a different set of NFPs depending on its function; an ISO 9126 standard [47] provides a list and a number of taxonomies for NFPs. A banking system may need security to be specified and measured whereas blog systems may not need this property. Similarly, the extent of details to be specified for a non-functional property also depends on the domain of the project. For example, a government project will have more data flows and validation criteria with multiple login requirements compared to a single sign on system; therefore the security or reliability specification will be in great detail for a government project compared to single sign on system.

As mentioned above that a single non-functional requirement may be handled by specifying more than one NFPs, the level of specification for non-functional property may need classification or categorisation of NFPs. There are different classification schemes for NFPs [67] and each classification scheme arranges them in a hierarchy. For example, the NFPs response-time, execution-time and throughput can be put in one category of performance and similarly the usability category can contain properties such as ease of use, effectiveness and time to learn. In the next section we will discuss a classification scheme that can be used for NFPs. The classification of NFPs is helpful in finding the association of different properties with the same functional component of the system and the dependency between NFPs. There are many proposed classification schemes for non-functional requirements with reference to NFPs. The classification presented by Sommerville [97] suggest that the classification schemes of non-functional requirements can be applied to NFPs because NFPs are derived from non-functional requirements. This classification scheme highlights high level classification of non-functional requirements covering not only measurable properties but also external, ethical and standards related requirements. The classification scheme discussed in [1] puts the NFPs into categories, which were also identified as Product non-functional requirements [97]. Another study [19] refers to classification of non-functional requirements and its lists the NFPs in the classifica-

tion that are derived from these requirements. All these classification schemes refer to non-functional requirements and their association with NFPs and this thesis will give a brief insight into the classification scheme [97] used in Zschaler's framework [110] because the QML/CS language is also based on the same framework.

Sommerville [97] identified three main categories for non-functional requirements as Product, Organizational and External requirements. The external requirements cover the ethical, legislative and regulatory requirements that cover NFPs that can not be measured from a software point of view. The organisational requirements refer to environmental, operational and development requirements of the system that are to be fulfilled by the organisation for smooth implementation of the system. The product requirements cover mainly the measurable requirements that are more closely associated with the functional components of the system and implemented by the component developer when developing the component. The NFPs that can be covered by product requirements are related to security, dependability, efficiency and usability. There are many NFPs that can fall under one of these categories; for example, the efficiency considers the performance of the system with consideration of the required and available resources. As discussed in [97], the NFPs that can be derived from the efficiency related non-functional requirements include but not limited to time behaviour, performance, throughput of both software and hardware components and their resource consumption so that we know how the system will behave in a specific working environment.

2.3 Software Language Engineering

Many of terminologies and formalisations for quality modelling language of component-based systems (QML/CS) developed for this thesis have been built on the basis of software languages engineering and meta-models. The following subsections explain the essential notions and concepts of software language engineering (SLE), model driven engineering (model, meta-model and code generation)and meta-modelling standards.

2.3.1 Introduction

Software language engineering is an approach to develop modelling and programming languages and therefore a modelling language can itself be considered as an artifact. The description of software language artifacts is crucial because it plays an important role for various tasks. Having a language description as a form of input helps the language builder in the process of constructing a set of supporting tools to be used by the language user. In addition, the language user can also benefit from language de-

scription to understand a program or model of the language. A language description can take two forms one of which is a BackusNaur Form (BNF) grammar while the other is a meta-model. Kleppe [57] pointed out that languages can be described by a meta-model which express the language's abstract syntax. In this thesis we develop QML/CS using the meta-model approach. The definition of the language based on a model uses object-oriented modelling to define the abstract syntax of the language, which is a meta-model for the language being created. This model should not only define the relationship of language elements but also separate semantics from the abstract syntax. There are many integrated development environments like Eclipse [99] that support meta-modelling of the abstract syntax as a meta-model using constructs including Ecore, which has information of the defined classes of meta-model.

2.3.2 Model

This subsection provides a definition and overview of what a model is and how is it related to the definition of a formal specification language to specify NFPs of component based systems. There are various answers to the question: what do models mean? Ed Seidewitz believes that *"a model is a set of statements that can be use to describe or specify some system under study (SUS)"* [93]. Since the model represents a system, the correctness and accuracy of the model specification can only be assured if the claims and statements made in the description stand true for the system under study. The consistency of the statements made in model description with the system being studied is also important to accept the model as a representation of the system [73].

An alternative, established by Jackson concerns the nature of the truth of the model stating that *"the Model is not reality"* [50, 73]. Meanwhile, Hesse considers the double rule of models in software engineering depending on when it is done,i.e, before creating the original system or described after the original system has been created. Selic [61] shares this view with Hesse that the models can be either descriptive or prescriptive; elaborating further that the model can be either created and used for developing a system or done as a description for a system that has been already created. Selic [61] adds to this viewpoint indicating the model can also be created during the development of the system so that behaviour of different components and their interaction can be understood. Fowler [38] has a perspective that the model can be linked with the stage of development of a software life cycle and the three levels are recognised including conceptualisation, requirement and design specification and the implementation.

2.3.3 Meta-Model

This subsection introduces the concept of meta-model and how it can represent different models and their integration. It also provides information about model-driven engineering and architecture and explains their link with a meta-model specification. It then talks about the mapping and transformation between different models so that interaction between different models can be formally specified.

The meta-model characterises the structure, semantics and constraints for a group of models. A precise definition of a meta-model is

> *"A meta-model makes statements about what can be expressed in the valid models of a certain modelling language"* [93]

It can be considered a model of a model, and is widely used in Computer Science, with several different meanings. The literal meaning of *Meta* is 'after' in Greek [74]. The aim is to generalise across different models to devise a model linked to a set of related models. A meta-model, provides the grammatical rules for the modelling language itself [74].

Modelling plays a major role in software development because it can demonstrate a system and helps in understanding on how the system is working or the way it should work. With the advancement in technology, the complexity of the software systems has also increased and therefore the Object Management Group [76] keeps working on the proposals to encourage use of a standard technique for modelling. There are many modelling standards documented in Model Driven Architecture (MDA) from OMG [76] for which the architecture is shown in Fig 2.6. Different design and implementation approaches [2], [17], [55]and [12] support a model driven approach to develop software systems where model is not just a conceptual entity and they are part of system visualisations created to describe the functionality and behaviour of the system. The modelling languages like UML help the application designers to map the concepts of a real system into a software model so that behaviour of different entities in the system can be modelled and their mutual interaction is understood accurately. Although modelling languages like UML are generic in their nature and can cover many domains, some domain specific languages are also created to model specific systems where notations for those systems may not apply generically.

2.3.4 Model Driven Engineering (MDE)

Bezivin and Gerbe [18] tried to define the modelling and model-driven engineering in different ways. They realised that researching model engineering is a potentially

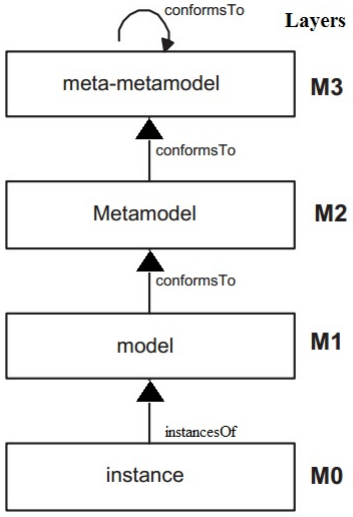

Figure 2.6: The four levels meta-modelling architecture of MDA as defined by OMG, derived from [77]

significant area because model driven approach should be considered as important as object-oriented approach. They also mentioned that models could potentially supplement, if not replace, the status of objects and classes. Bezivin worked with Breton [16] in 2004 to advocate that model-centric approach is comparatively more suitable than the object oriented approach. There is lot of focus on the aspect and perspective of the system from its implementation and usage point of view and therefore models with high abstraction and low granularity may be a better choice than equivalent object-oriented models. They also mentioned that a meta-model would work as a guideline or standard for the model to conform to and therefore representing the specific aspect of the system. MDE considers making models first-class citizens of software development. It also deals with the research that works on finding new ways to implement transformational techniques so that effective models and their mapping can be created to facilitate the transformation.

The different standards of OMG facilitate model driven engineering but each standard covers a specific perspective of model driven engineering. Meta-Object Facility (MOF) provides a type system along with set of interfaces to create and manipulate types. UML is a modelling language that provides structures and notations to model behaviour of different entities, components and their interface between those entities and components. The Common Warehouse Meta-model (CWM) covers specification perspective of modelling metadata so that different warehouses can interchange and exchange information. The QVT (Query/View/Transformation) helps with transfor-

mation between different models. The XML Metadata Interchange (XMI) [80] helps in using XML data format for exchanging meta-data so that it is easily understandable and implemented by transformation tools. Java Metadata Interface (JMI) helps in using Java for creating and manipulating modelling languages like UML.

2.3.5 Model Driven Architecture (MDA)

Model Driven Architecture (MDA) is an approach to controlling complexity, accomplishing high levels of re-use and decreasing the developmental effort necessary for software development [81]. MDA has been defined by the Object Management Group (OMG), as a framework that provides an emerging collection of patterns and technologies focused on a particular software development style [81]. Furthermore, it essentially expresses the relationship between models and their mutual transformations. MDA gives support for transformations between platform specific and independent models (PSMs and PIMs) based on their relationships and use transformation techniques to link them. Platform-independent and platform-specific models of MDA are not relevant in the context of this thesis, thus these concepts will not discussed.

In terms of the infrastructure of MDA, primary standards have been defined by OMG: UML; Meta Object Facility (MOF); XML Metadata Interchange (XMI) [80] and the Common Warehouse Meta-model (CWM) [77]. There are four layers of modelling associated with OMG standards, and it plays an important role in the MDA framework. OMG defined the four layers of modelling for its standards represented in Fig 2.6. These layers can be classified as follows:

- M0: As in Fig 2.7, the instances of the M0 layer represent the actual instances of the running system. For example, names or addresses of people "Abdulrahman" in "King's College London". This information can exist anywhere in the system such as the database.

- M1:As in Fig 2.8, the model of the system which classifies and categories the instances of layer M0, means there is a relationship between M0 and M1.

- M2: As in Fig 2.9, the model of model are the elements that exist in M1 layer are basically instances of classes in M2 layer. In other words, each element in M1 is an instance of M2. This layer is called a meta-model.

- M3: As in Fig 2.10, the model of M2 is almost similar to the relationship that exists between layers M0 and M1, and it is higher layer, in which the elements of M2 layer can be instances of M3 [100].

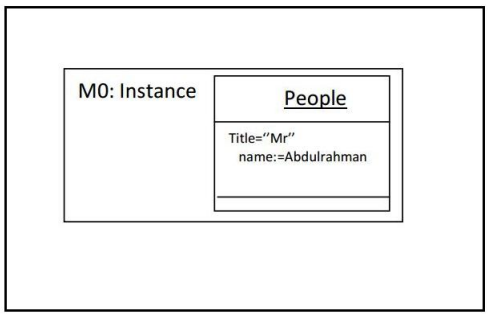

Figure 2.7: M0 Instance derived from [100].

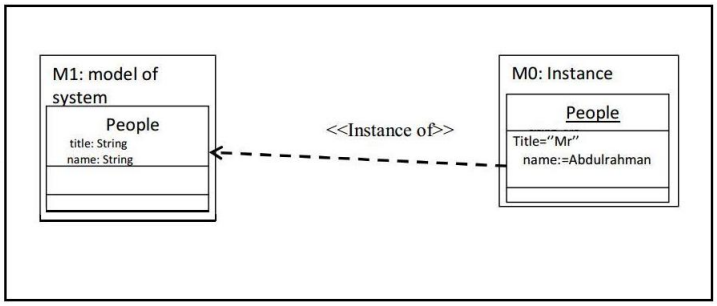

Figure 2.8: M1: Model derived from [100].

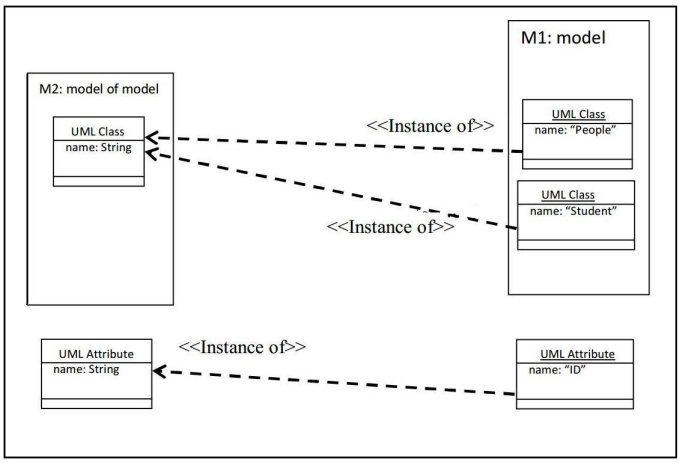

Figure 2.9: M2: Model of model derived from [100].

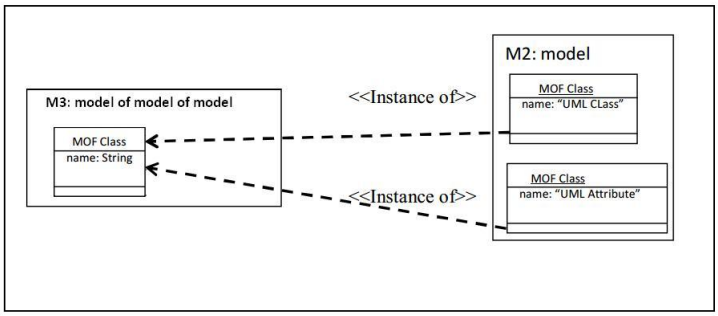

Figure 2.10: M3: Model of model of model derived from [100].

2.3.6 Model Transformation

Model transformation is an automatic way of generating target model from a source model [56]. Model-to-text (M2T) transformation is considered as an important approach of model transformation, which can be used to generate codes from models [65]. Among others, Model-to-text (M2T) transformations is supported by the Epsilon Generation Language (EGL) [65], which can be used to translate the content of a model from one language to another. In this thesis, EGL is used to translate QML/CS specification into TLA+ specification in order to show the semantics of QML/CS based on the transformation template.

2.4 Related Work

This section will discuss related work in the area of specification languages for specifying NFPs (NFPs). In the current literature, a number of languages and frameworks are proposed for modelling NFPs. These approaches address the problem of formal specification of NFPs but most of them are limited in their application. Some of the languages and frameworks are generic in nature but have Low-level formalisation, which makes them practically unusable as in [3, 9, 39, 40, 53, 87, 110]. Their practical usability is constrained by lacking the syntax and semantics to be able to specify NFPs completely. There are some other languages [61, 78, 79] that are generic but are not suitable because of the limited support provided by them in specifying component-based systems. Also, some of the languages [11, 13, 20, 42, 58, 94] provide a formal specification of NFPs but they offer limited support for NFPs, covering specific properties like performance and reliability.

As mentioned earlier, Zschaler proposed a framework which is generic, having a strong formalisation and claims to propose a formally defined language, named quality modelling language of component-based systems(QML/CS). However, this claim does not

corroborate the reality as QML/CS is not formally defined rather it is limited to only few examples of how the language should look like. QML/CS is only discussed in [110], and no definition for this language was proposed nor a grammar. With the information available in [110], it is challenging to define a language because there is no grammar, formal specification and parser tool that supports the specification. It also is not obvious how will one construct such a language, more details of these shortcomings of the proposed QML/CS will be discussed later in Chapters 4 and 5. Furthermore, it also has no tool that can be used to write specification based on this language and his research paper [110] does not address the challenges of building a formal definition of the specification language because there is no practical implementation. This thesis has defined and developed a new specification language that is based on the Zschaler's framework and it takes inspiration from proposed QML/CS examples in the paper. It also addresses the challenges of building a formal specification language for NFPs of CBSE.

This chapter will discuss the above mentioned languages based on the categories which have already been identified and will explain how they work. The criteria for associating them with a category and linking it to the gap they have with current work will be mentioned. The basic criteria parameter for categorising the languages is based on their formal semantics, genericity, practicality, application domain and complexity. The categorisation of the languages and frameworks is presented in table 2.1. Each language will be discussed separately to explain its attributes and why it ended up in the assigned category so that its gap with new QML/CS language can be clearly identified, thus, this section will evaluate each one on the same categories and summarizing it in the form of a table.

2.4.1 QML

QML [40] stands for 'quality of service modelling language' and was designed with the consideration that inclusion of specification of non-functional property at design time is important because that is when architecture and context of the application is defined. It also takes into consideration the context of the usage of an operation or service so that the non-functional property can be defined at individual operation level or the service level as a whole. It uses three key concepts of contract type, contract and profile. The contract type identifies the type of non-functional property domain that is being addressed like reliability and performance and it mentions an abstract representation of the structure that is used later to define the property at a discrete level. The contract is discrete realisation of the contract type and specified as instance of contract type and provides detail of different dimensions of the non-functional property being defined. The profile property handles the association of the contract and contract type with a specific service or operation, which is made

possible by QML's design capability that allows to specify NFPs at object-oriented concepts level like classes, interfaces and objects.

Although QML supports multiple profiles to be defined based on the context of the usage but that has to be done at design time and that means it uses a static adaptation approach to different contexts. It makes QML usable for only design time association of NFPs with classes, interfaces and operations. Although QML makes use of QoS Runtime Representation to dynamically manipulate NFPs buts its scope is limited and does not cover object configurations, making it less suitable for CBSs. It helps in defining different perspectives of the each dimension of NFP but it does not specify the impact of each property on the service or component being targeted. It is clear in its usage because the contract type and contract use easily readable structure to define the properties but can be sometimes ambiguous especially when defining values for different dimensions of the same property. It does not mention if the definition at profile level can overwrite definition at contract level. It is practically usable for design time QoS association and manipulation and it virtually can define any NFP.

2.4.2 NoFun

Xavier Franch designed a language named NoFun [39] based on ISO/IEC quality standards. A hierarchical attribute specification for NFPs is proposed. Three key concepts are used named *non-functional attribute*, *non-functional behaviour* and *non-functional requirement* that specify different aspects of a non-functional property. The non-functional attribute defines the non-functional property being specified, non-functional behaviour defines the value assignment and non-functional requirement defines any constraints associated with the non-functional property when associated with a module or component. The non-functional attribute has many characteristics that cover domain of implementation giving expected behaviour of operation or component, type of NFP to be simple or derived, scope of association on which components it is associated with, number of different definitions it may have for different contexts and whether it is defined for an individual operation or the whole component.

Although the language NoFun claims its application for both component-based and procedural programming based systems but it needs adaptation for both to start using NoFun as their specification language for non-functional attributes. It has defined some notations to avoid natural language description but those notations miss clarity in their meaning and definition of too many notations and their reuse in different contexts makes the language ambiguous. It can be suitable for component-based systems but the NFPs in component-based systems are derived based on individual operations of that component. Therefore, dependency on internal architecture of the component is higher because specification needs knowledge about internal structure

of the implementation. It claims to have well-defined semantics but there is no evaluation strategy discussed that may indicate the level of formalisation of syntax and semantics. It also means that although they have many concepts for syntax and semantics but their formal structure is not well defined.

2.4.3 CQML

Aagedal introduces Component Quality Modelling Language (CQML) [3] that is a language to specify NFPs of a system. Compared to QML, CQML does not dictate how the functional specification of the system should be written and specification of NFPs is independent of functional description of the system. It can be used to describe NPFs generically. It allows definitions of basic data types (e.g, Number, Set), simple properties and derived properties. However, derived properties are limited in their definitions, which means that they can be constructed from either extending existing simple properties or the composition from other properties with no indication of how they are composed. CQML consists of four types of specifications for constructs. First, QoS characteristic is classified as a basic construct, which represents a single measurement. This QoS characteristic is mainly composed from a characteristic's name and its data type. QoS statements are the second construct, and are mainly used to restrict each element of QoS characteristics to define specific ranges of values within a single QoS statement. QoS characteristics and QoS statements concentrate on the specifications of the independent QoS, and describe how the interface actually is and how the QoS mechanism is implemented. The third construct is QoS profile, which involves several QoS statements as specific components. The fourth construct relates to QoS categories, which are used to join the three constructs above.

This language is designed for component-based systems but its focus is more on the syntactic level and there is no formal specification for semantic elements. The language does not have clear specification for some of its concepts like *Flow* and *EventSequence* and semantics of concepts change in different contexts making their use ambiguous. The confusion and informality in semantics makes it less suitable to use because their meaning will change from concept to concept. Although it is generic in the sense that it can define any non-functional property but it is not generic for any component-based system because specification has tight coupling for component and system specification of NFPs.

2.4.4 HQML

The Hierarchical QoS Markup Language (HQML) [42] is a XML-based specification language that helps in specifying NFPs with emphasis on discovery of suitable service on World Wide Web. The NFPs specification is saved as a HQML file and the client

can request a service by passing the requirement, which is then compared with HQML files available for each service that can be used. The user, who requested the service with pre-defined QoS requirements, is presented with services for which the HQML specification matches the user requirements and the user makes the decision on which service to use based on the context of requirement. Although the language uses XML as a specification language and XML is widely accepted language to exchange information, its focus is more on searching the suitable service based on the required QoS parameters. It makes it typically suitable for cloud environments where users can request a service based on the requirement and the cloud is able to serve with the service that matches those requirements. It makes this language more suitable for a negotiation protocol than used in modelling because it is meant to be used to exchange the information about NFPs and matching with the service specification HQML file. This also indicates that it is not suitable for component-based systems because it is more aligned for a user demand-based service applications over WWW.

The implementation of this language based on XML makes it generic in the sense that it can be used to specify any non-functional property because XML syntax allows it to define any structure with attributes and elements. The only limitation XML puts is that names of the elements or attributes can not start with a numeric value and therefore all the NFPs, their attributes and constraints must follow this syntax. It is more like a configuration language that is used to create QoS configurations for a system that can be used to compare with user demands to propose suitable service. Although it is practically usable language but its syntax is based on generic XML and therefore it can be ambiguous because XML allows to specify the same property in two different ways. Also it is not designed to be useful for every type of system and its practicality is limited to the WWW driven service based systems.

2.4.5 CQML+

CQML+ [43] is an extension of CQML [3] is a specification language for NFPs of component-based systems. It was developed as part of a larger project called Quantitative properties and Adaptation (COMQUAD) [43] that focused on creating a system architecture that can help development components with clearly measurable NFPs. CQML+ develops quality characteristics aligned to measures issued by the measurement designers. Quality characteristics comprise a name, a semantic and a domain. The values clause offers quality characteristics of properties being specified. Further, the CQML+ offers the construct of quality profiles, which assist in associating functional property specifications and non-functional specifications of properties. It implemented a UML based meta-model that allows computation to be part of the specification so that the specification can be made independent of the running system containing the components. It also offers a new meta-model for specification of

resources so that their demand and availability can be made part of the specification.

Since this language is based on CQML so it carries the problem of semantics and ambiguity along however it addresses few of them. The unclear mapping between offered and used QoS of a component in CQML was solved by proposing a new relationship clause so that ambiguity in specification can be reduced. It also introduced *tuple* type to handle structured NFPs so that more complex type of NFPs can be specified. New meta-models for resources, mapping of use and computation specification make it more practical than CQML and add the capabilities of specifying other constructs in the system like resources and what are expectations of the component from those resources. It is comparatively better in being practically usable than CQML but it did not address all ambiguity issues in CQML. It is suitable for component-based systems because it defines the use and expectation of NFPs between the components and make this definition independent of the running system.

2.4.6 SLAngs

SLAng [61] is an XML-schema based specification language that was primarily designed to write service level agreements (SLAs) including specification of QoS attributes. It covers the negotiation between two entities such as component-to-component, service-to-component, container-to-component, container-to-provider and similar communication in a distributive integration environment where services and components from potentially different providers work together to provide functionality of the system. It considers the different qualitative and quantitative requirements of QoS specification in different contexts and the XML-schema helps in defining different specification requirements depending on the type of the project or domain being targeted. This language claims transformation advantages but it is XML that supports transformation to any format as needed using many tools like Eclipse and XSLT implementation to convert XML-schema to any format that may be needed. XML-schema also makes the integration of this language with existing XML based schema languages like WSDL [102] that work as a contract to use the web services.

The language mentions about monitoring, validation and enforcement of QoS attributes but there is no clear specification of how the language can help in it because it is merely based on XML-schema itself. They have defined some domains like component, services, persistence etc that they specifically supported in their original specification but XML-schema is generic in the sense that it can be used to define any structure of information that may be needed for QoS specification. So SLAng derives this from XML-scheme that it can be used to specify any non-functional property but it does not cover how an existing or new architecture can integrate this and what changes may be required; this makes it less practical because it apparently requires

architectural changes in the implementation to support this. Its complete reliance on XML makes it less formal in its semantics because it did not mention any controlling infrastructure that can be used to restrict the names or attributes of non-functional property and it is left to application designer or component developer on how they specify their system and implement it.

This language was improved in [95] where they used UML to model the language so that formal semantics can be defined and the language can be made more practical as UML case tools can easily support the specification once it is modelled in UML. However, low formalisation of UML itself keeps SLAng less formal as well. They used abstract syntax definition of UML defined by precise UML group and then associating it with different context domains so that semantics for that domain can be defined for application to that domain. The generic nature of XML and modelling it with UML structures expands the use of this language in component-based systems, provides formal semantics but it still lacks the information about integration effort required for working with existing or new systems. The UML modelling makes it less ambiguous when used for specification because it makes the abstraction specific for the use but that also means that a lot of specialisation can exist for the same abstraction that may give different meanings to the same concepts.

2.4.7 UML SPT and MARTE

The UML SPT profile [78] is a specification profile from object management group about the meta-model of non-functional attributes named performance, scheduling and time. It is based on UML [82], which is a powerful modelling language that allows users to specify a software system. The performance, scheduling and time are modelled as UML models so that any component or system that confirms to UML specification can use this as specification standard. It did not consider specifically component-based systems and provided a generic specification and there is no clear restriction on its application to any domain or type of project.

These non-functional attributes are concerned more with operation or run-time of the system and are very important for real-time systems but UML SPT has some limitations in handling them for real-time systems. Many problems have been identified [84] and reported to OMG so that specification can incorporate those changes such as SA Profile issue and TVL notation. It is not generic because it handles only three aspects of NFPs and it does not address challenges of component-based systems specifically as well. It is less formal because it is based on UML but integration with UML case tools extends its usage in specification for systems makes it easy to use.

The problems identified in UML SPT profile for real time systems were addressed

and solutions provided in MARTE [79], which is another specification from object management group and it adds support for performance, schedulability and time specification to be used in real-time embedded systems. So it is primarily an extension of UML SPT profile and offers all the advantages discussed above. It provides a common technique to model specification of software and hardware components in the system so that software and component developers have a mutually understandable specification. This common specification increases the interoperability between different tools in the development life cycle because they all use the same specification and therefore transformation of specification into development is easier and so is the evaluation once implementation is completed. It also facilitates construction of models that can be used to measure performance, schedulability and time in real-time embedded systems.

2.4.8 CB-SPE

Different Frameworks are also proposed in regards to improving QoS in CBSE. The CB-SPE is one such framework that is a generalization of the SPE approach to CB systems. The CB-SPE bridges the levels between the component developer and the system assembler. This approach, outlined by Bertonlino and Mirandola (CB-SPE) [13], is based on the UML SPT profile [78] and uses XMI [80] for data exchange. It extends SPE approach [96], which was written for component-based systems to be able to support design and specification of standardising the component development. The tool is not developed from scratch, but draws on several other freely available tools, for instance using ArgoUML for UML processing, to perform its functionality. These modules, mostly based on other available free tools, are joined together and XMI is used to transfer data between them, and thus the output of one module becomes the input of another.

With Component-Based Software Engineering (CBSE) as the base paradigm for this tool, alongside the emerging importance of component-based development in designing reusable component-based systems, this tool provides a very good initiative to model the system using CB-SPE. It implements the CB-SPE framework for Software Performance Engineering and addresses the problem of component coupling by considering the components as independent functional units working together based on agreed integration standard and QoS requirements. It provides input notation to support the specification and evaluation of NFPs making the SPE approach simple to use and also derives NFPs of the system from the QoS attributes of the component. It is not generic because it can specify only limited NFPs and the design focus was also on embedded systems. Ambiguity is also contributed by its basis on UML based techniques and tools and that also results in less formal semantics.

2.4.9 Robocop

Robocop [20] is one of the component-based frameworks that works as a middle layer to enable the development and extension of component-based solutions so that the re-use of the components can be ensured and their integration with other software and hardware components can be facilitated. It was designed to enable changing consumer devices or services to incorporate new requirements easily into the system so that the time-to-market can be reduced without compromising the robustness of the system. This is a framework that provides support for analysis as well as development techniques, infrastructure to support the integration and specification of NFPs like robustness, reliability and resource consumption. These NFPs are primarily suitable for devices and services where frequency of change is higher and time to complete and integrate has to be controlled. The framework provides specification of NFPs robustness, reliability, resource consumption and also provides techniques that can help to validate and predict these NFPs.

Although Robocop provides an option to specify NFPs but that is just a part of the bigger framework and it is done only to support the framework capabilities. The components and applications developed under this framework are allowed to use the techniques to check their NFPs. The framework also exposes the measurements for the NFPs to the user so that it can seen which component or application is satisfying the constraints determined by them. However, there is no formal semantics to specify them and the framework does not allow specification of most of the NFPs and is limited to robustness, reliability and resource consumption. This makes the framework non-generic in not only support for a variety of NFPs but also the type of systems it can work with as it is closely targeted for component-based systems in rapidly changing consumer devices market. Although it is for a specific domain but it is designed around component-based systems and therefore solves the problems of components working alone or with other components in the system and define their integration protocols so that new components can replace old ones. It is practically usable but the scope of its application and limited support for NFPs make it less likely choice for application to a larger domain of projects.

2.4.10 Zschaler's Framework

A formal semantic framework is presented in [110], which allow the specification of NFPs for a component-based systems. It is defined on the basis of an extended temporal logic of actions (TLA+) and derives the non-functional property of components and services from these measurements and it is focused only on components that have available or determinable NFPs. There are several advantages to using TLA+, such as the fact that they are easy to use, various ranges of properties can be expressed

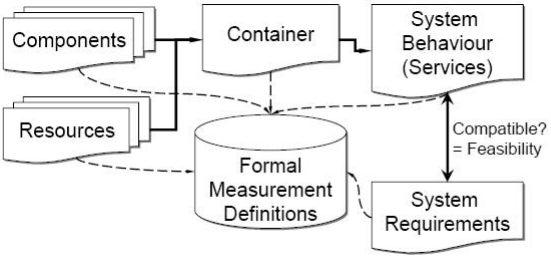

Figure 2.11: the specifications derived from [110].

and proof rules with model checking analysis techniques can also be supported. The framework provided five types called service, component, resource, container and measurement to represent its core concepts and they can be used to specify different levels of abstraction required to formally specify NFPs of component-based systems. The concepts of service and component complement each other in a way that service is the interface provided to use the functionality implemented in the component. The concept of resource is defined so that the component and service can be associated with the required resource and the requirement as well as availability of the resource can be specified formally. Rather than defining these concepts separately, a container is defined that can be used to join a component. the service it provides and the resources it may need to perform the task. The concept of measurement is defined to specify NFPs that is linked with either a service or a component and is used to mention the constraints for the NFPs as well. Fig 2.11 presents an overview of the specifications outlined in Zschaler's approach [110]. There are two major sides to establishing component-based systems with the described NFPs:

1. Component developers to specify NFPs of component in such a manner that it has certain constraints attached with it.

2. Application designers and the runtime environment must employ these elements so that the NFPs required from the application can be assured.

To provide a clearer picture of key concepts and types of specifications, the system model presented in [110] can be viewed in the Fig 2.12. Zschaler [110] illustrates that a component-based system is composed of a container that uses components and resources, provides services and has a container strategy. As mentioned earlier, it uses TLA+ to add formalism to the specification of NFPs and the focus is on specifying them in a way that system level NFPs are derived from component level specifications. The framework considers specification of all NFPs with emphasis on measurable properties so that expressions for their evaluation are also expressed part of the specification. Also the design constraints limit the ability of this framework

and any languages based on this framework to be able to specify the properties that can not predicted or measured statistically.

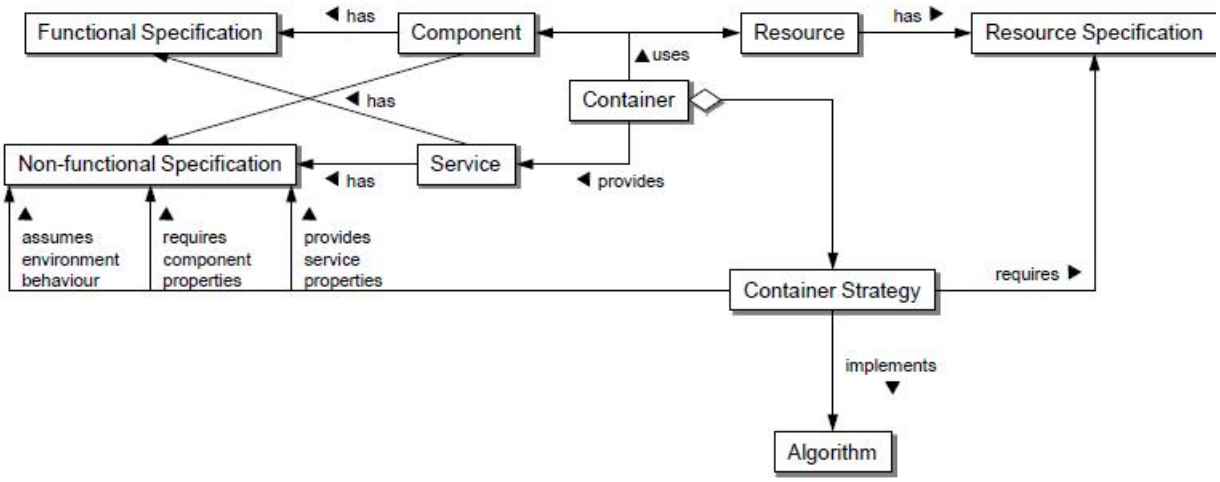

Figure 2.12: System model derived as a UML diagram [110].

QML/CS

'QML/CS' stands for quality modelling language for component-based software. It is a specification language for NFPs and considers two main concepts of the architecture including component and service. It mentions about how resource demand can be specified as part of NFP specification and presents ideas for integration of resource with measurements to give an overall representation of the system. The QML/CS language, as it exists, is not well defined, and shows only general principles of how the formal specification may look; therefore the development of a new formal specification language that is well defined is an important step towards building a practically usable formal specification language for NFPs of a component-based system. As it aims to handle measurable NFPs of component-based systems; first of two main concerns is to handle for a practically usable language would be to allow the application designer to specify NFPs in measurable way. The second concern is that the specification should be complete enough so that the component developer can use that specification to implement NFPs in the system. This implementation should allow the evaluation of NFPs against the specification.

The low level formalization of Zschaler's approach, as it is based on TLA+, makes it unusable for realistic application because there is no formal language that can be used. Also the presented concepts of service, component, resource, container and measurement are not complete and do not cover all dimensions of the NFPs being

specified. We aim to remove this Low-level formalization and replace it with high-level abstraction specification so that it can be applied for realistic software systems built from component-based architectures.

2.4.11 TADL

In 2008, Mohammad and Vasu [72] developed TADL, which stands for architecture description language for specifying trustworthiness of the system at architecture level. It is very specific to architecture description and is not only designed for NFPs. It has more formal description of the services, contracts needed to use those services and support of those contracts at interface level. One of the advantages of this language is that it specifies the data parameters as well so that the control on the content and type of content is established. This serves the purpose of this language for being able to specify and evaluate NFPs related to trustworthiness like security, safety, availability and reliability.

The language is not generic because it does not address any other properties but security, safety, availability and reliability. Also NFPs are just a subset of the specification abilities and that too is limited; this makes this language less practical. It is suitable for component-based systems because the architecture it supports for architecture description is well suited with component-based architecture. The language does not have any formal semantics and there is no tool that supports specification using this language.

2.4.12 E-Motion Observers

Troya and Vallecillo [103] present an approach similar to Zschaler's framework [110], which is considered to be an extension of their e-Motions environment for the development of domain-specific languages. In-place model transformation rules are used to define the behavioural semantics of these languages. The additional 'Observer' objects are then added, in order to define NFPs and the basic semantics rules are extended by updating the values of the observer objects.

However, the approach of Troya and Vallecilla is different from Zschaler's semantic framework in a number of ways, such as it requires a set of rules for the target system to be modified in order to add the semantics of updating observer objects. Furthermore, Duran, Zschaler and Troya later introduced a new approach i.e. integrating the previous approaches which resulted in allowing the modular specification of observers in e-Motion [35].

Observers in e-Motion involve a different approach, addressing the problem of validating the behaviour of any given operation conforming to the expected measurement specification parameter; the same problem will be solved in this thesis using the weaving model technique. The main difference between the two approaches is the way the behaviour of the context model and the application model is expressed. The E-Motion Observers use a graph transformation method to express this behaviour; by contrast, the solution proposed in this thesis uses a state machine model to express the behaviour of both the context model and the application model. The E-Motion Observers model considers the transitions to be first class citizens, not the states and states are instances of their graph types meta-model. It specifies transitions by defining rules, with each rule relating to a potential transition. This means that different rules are defined for transitions of a context model and the application model and therefore these rules will need to comply with one an other when we map a context model to an application model and vice versa. The E-Motion Observers model is a good example where the source and target rules are different but they merge to provide a complete specification; although this tight coupling of the source and target rules makes the observer approach less reusable.

The observer model in [35] somehow implies a mixed or merged definition of a measurement, integrated with a context model, whereas in this thesis both measurement and the context model will be treated as two separate concepts. The state-machine model is comparatively simple than the graph transformation as it can be represented with simple states and flow between them where as graph transformation needs mathematical notations to represent it effectively. The states of a state-machine model provide more insight into the internal behaviour of the method and therefore an improved accuracy of measurement can be sought. The approach in this thesis makes the concepts of measurement and the context model more independent and loosely coupled. It makes definition of measurement easier with possible transformations to more than one context models based on the context in which the measurement is being used. This also helps in making the measurement definition and then mapping it to a context model based on the environment.

2.4.13 Palladio

The Palladio Component Model is the base for Palladio-Bench [11]. PCM is composed of four main models, which express different aspects of the system, and a usage model that expresses user behaviour. The UML-like graphical editor is used by PCM to model several performance analysis methods for software systems. Palladio, based on the Palladio Component Model is an advanced performance modelling language. It supports the meta-model in relation to quality of service attributes such as performance, reliability and maintainability of software systems based on component

driven architectures. PCM was also used as a base concept from which several core elements of Descartes Modelling Language are derived [58]. Component, System, Resource Environment, Allocation and Usage are sub-models within the PCM model, and PCM uses a parametrised approach to control the behaviour and influence of these modules in a meta-model. This parametrised approach gives the designer more power and provides an opportunity to identify performance related problems early in the software development life cycle, thus increasing the chances of good quality service software systems.

Although PCM provides for quality of service, it supports limited NFPs like performance and reliability. The formal specifications of performance and reliability are non-generic, and therefore cannot be used to specify other NFPs, such as response time, scalability, correctness. As a result, this makes application of this language suitable for only a few NFPs. It has formal semantics but that does not help in making it practically useful because its support limited to specific NFPs mentioned above. A recent paper [34] discussed the improvements in the way PCM handles specification of NFPs but it is too early to say that it handles the existing limitations mentioned in this thesis.

2.4.14 ProCom

ProCom [94] is a component model designed for control-intensive real-time embedded systems to help develop component-based solutions where controlling of the behaviour of the system in the context it works is very important. ProCom is a comprehensive component-model that has strong semantics to be able to specify the requirements and design objectives of the system being developed and then provides guideline for the whole development process. It also covers the testing and evaluation of functional and non-functional aspects of the components and define how they can work together to provide the system functionality. It follows the PROGRESS approach [22] because it covers the three distinctive processes like design, analysis and deployment covering the whole life cycle of the system from inception to deployment. It provides a hierarchical specification technique to specify the non-functional requirements at different level of granularity so that each next level derives from the upper level and provides further information to specify.

Although the component-model has the ability to specify the non-functional attributes at different granularity level and suitable semantics are available to specify but this multi-level specification makes it complex to specify and doing it without a tool becomes hard to maintain. Its practically usability is constrained with the requirement of a tool and the complexity of specification may render it less usable when suitable tool is not available. The multi-valued specification of the non-functional attributes

provides a comprehensive specification strategy to specify non-functional attributes, their values and evaluation techniques and can be very useful if there is a suitable tool that supports the specification. It is designed to work with component-based systems and its architecture support specification and evaluation of NFPs for components and their integration.

2.4.15 The Framework of Jezek and Brada

Contrary to many component-model frameworks [20, 94] where they propose a new component-model framework and then suggest a specification standard for NFPs, this framework [53] addresses a very important domain of facilitating existing component models and frameworks that miss specification of NFPs in their design. It provides a comprehensive framework to enable existing component-based systems to use NFPs so that the consistency of the components and the way they work together can be improved; and that is without changing the architecture of the existing component-based systems. It proposes a three step strategy to integrate the NFPs with existing components where each step is handled by a module in the framework. The modules are named as definition, attachment and evaluation of NFPs. A repository is maintained to define the NFPs and interface is provided so that these NFPs can be attached with the components. The evaluation module will ensure that NFPs are validated as per their definition when the components work together. Since the framework is designed independent of any native component model; the component-based system that will use it has to support the integration with this framework so that its components can use NFPs definition from repository.

Although the framework claims to be independent of the component model used to develop the component-based system, it still needs changes in the component-based system to work together and these changes will vary from system to system. This requirement is based on the assumption that the components should be able to integrate with the repository of NFPs so the component architecture can have an impact on the integration. This compromises the generic nature of the framework and also exposes the dependency of the framework on architecture of the target component-based system. The Zschaler's framework [110] addresses this problem by keeping the specification of NFPs separate from the implementation so that it can be used to specify NFPs without worrying about the underlined architecture and can potentially work with any component-based system. The framework [53] has formal semantics to define NFPs and it uses the concept of basic and derived properties like [39]. It also provides the semantics to integrate components feature wise so that the evaluation can consider the values attached with features of the component to check the binding of the components. However, this level of granularity makes it complex to use because the components will have to link with NFPs for each feature and the component over-

all. It provides different ways to specify the property and its specification is based on mathematical concepts of set and their use in building relationship between NFPs of two components being integrated.

2.4.16 Descartes

Just like Palladio, Descartes Modelling Language (DML) [58] is another architecture description language that provides modelling of non-functional requirements related to performance and availability of the system. It also provides resource management abilities so that the above mentioned NFPs can be ensured while the system is operational with emphasis on efficient resource requirement and management. It addresses the challenge of run-time resource management with changes in the working environment depending on dynamic changes in the requirement of resources. It is different from other languages and frameworks discussed here that it is the only one handling specification of run-time resource management along with specification of NFPs.

The specification of resource management and run-time management of the resources makes it very complicated to use because the resources are managed in two layers; virtual requirement of the resources and then physical resources allocation to meet the requirement. Although this language models quality of service, it offers limited support for NFPs, covering performance and reliability only. In addition, the formal specifications of performance and reliability are properties-specific, and therefore cannot be extended to specify other NFPs, such as response time, scalability, correctness and so on. This limited support and lack of generalisability makes application of this approach suitable for only a few NFPs. It is suitable for component-based systems because it handles the resource management for the components and the system at run-time. It has formal semantics to specify performance NFPs and their related resource management to dynamically adapt to changing resource requirements therefore the specification for both NFPs and resource management is detailed enough to handle the complications of the dynamic working environment.

2.4.17 The Framework of Banerjee and Sarkar

The latest advancement framework [9] aims at formal specification of extra-functional (another synonym used for non-functional) properties (EFPs) and it uses Z-notation [98] to specify the NFPs. It handles complete process of NFPs from definition to evaluation and evaluation is facilitated with first assigning the EFPs to components and then using ZTC type-checker [54] to evaluate the correctness of the assigned properties. Although the specification model was designed to be suitable for systems designed using Z-Formal Specification of Component Model (ZFSCM) [10] but it is claimed to be suitable for any component model driven component-based system.

This framework has lot of similarities with [53]. They both use the same concepts of simple and complex properties and are very similar in realisation of NFPs. This new framework even mentions the simple and derived properties and concept of composition to create or specify NFPs based on other NFPs. Additionally it relies heavily on ZFSCM and uses predefined and established Z-Notation concepts of ZFSCM to formally specify NFPs and assigning them as a property in the target component. This framework proposes a categorisation for NFPs putting them into directly composeable, architecture related, derived, usage-dependent and context-oriented properties but the classification is not clear as it potentially puts the same property in different categories.

It presents the good level of support for component-based systems and has an evaluation mechanism using ZTC type checker that ensures correctness of the specification. It is practically usable because it provides support for specification, assignment and then evaluation of NFPs but it is complex to use because the user will have to learn Z-notation, ZFSCM, ZTC type checker and the descriptive specification protocol. It is generic in the sense that it can be used for any type of component-based systems and also has the ability to specify virtually all NFPs; both simple and complex that are derived from simple property specification. Having offered all these advantages, this framework has a challenge to offer something new because it is very closely based on the concepts discussed in [53] and suggests another notation to represent the same concepts.

2.4.18 Discussion

This section will discuss related work in the area of specification languages for specifying NFPs. Current research of the specification language sector and the inspection of available languages is likely to raise the following observations concerning the requirements of a service-centric QoS specification language:

- It should be generic so that it can be used to define a variety of NFPs and a consistent specification can be provided.
- It should have strong semantics so that clear specification is written and ambiguities are minimum. Also each feature should have clear understanding from the semantics so that they can be evaluated and automated.
- It should be practically usable so that it can be used in software development life cycle to design and convey the non-functional requirements from inception to evaluation.
- It should be comprehensive enough to able to specify NFPs for component-based systems and other development models.

- It should be easy to learn and its integration with existing and new architectures should be smooth so that its acceptance can be increased.

The table 2.1 is presented based on specific criteria including genericity, practicality, application domain and complexity. Genericity implies whether the language is limited to a set of properties, that is NFPs for component-based systems. The practicality criterion relates to the ability of a language to be practically usable in specifying NFPs of component-based systems. Application domain criteria refers to the support provided by the language in specifying component-based systems and if it is specific to component-based systems or it can specify other type of projects as well. Complexity indicates how much learning is needed to understand structure of the language, its features and its use in target system.

Most of the existing specifications, as those presented in table 2.1, limited to specific NFPs (e.g., HQML [42] ,CB-SPE [13], Robocop [20], Palladio [11], TADL [72]). Others slightly support the necessary concepts for NFP of component-based system (e.g., QML [40], SLAng [61],UML SPT [78], MARTE [79]). The approaches (e.g., CQML [3], CQML+ [43], Zschaler's framework [110]) are generic but their low level formalisation makes them practically unusable. It is desirable to design a language that is more readable, can hide the complexities in abstraction and allows formal specification of NFPs practically. E-Motion Observers [35] is a framework that is generic and practically usable but it is very complex because it combines the concepts of context models and measurements; and the concepts are tightly coupled. As discussed above, although Zschaler claims to propose a formally defined language, named quality modelling language of Component-Based Systems(QML/CS). However, this claim does not corroborate the reality as the framework does not feature a practically usable language, and the proposed QML/CS language, as it is discussed in [110], is not well defined, referring to general principles regarding how the formal specification might look [52]. We therefore propose a generic usable high-level specification language for NFPs of CBSE.

2.5 Summary

This chapter has reviewed related research, including role of MDA in Application Development, NFPs, Development Process for NFPs, quality modelling languages and approaches. Moreover, the review of literature and related works supports this research in helping identify gaps in modelling quality language generically in component-based systems and guide direction to the proposed quality modelling language in as an actual language, systematic approach. The following is a summary of

QoS approaches	Genericity	Practicality	Application Domain	Complexity
QML [40]	Yes	Yes	General	low
NoFun [39]	Yes	No	CBS	low
CQML [3]	Yes	No	CBS	high
HQML [42]	Yes	limited usage	Web Service	low
CQML+ [43]	Yes	No	CBS	low
SLang [61]	Yes	Yes	General	low
CB-SPE [13]	No	Yes	CBS	low
UML-SPT [78]	Yes	Yes	General	low
MARTE [79]	Yes	Yes	General	low
Robocop [20]	No	Yes	CBS	low
NFPCBF [110]	Yes	No	CBS	high
TADL [72]	No	Yes	CBS	high
Palladio [11]	No	Yes	CBS	high
ProCom [94]	Yes	No	CBS	high
GEFPF [53]	Yes	Yes	CBS	high
Descartes [58]	No	Yes	CBS	high
E-Motion Observers [35]	Yes	Yes	CBS	high
EFPCBSF [9]	Yes	No	CBS	high

Table 2.1: Comparison of QoS Specification languages and Frameworks

the main issues in this chapter.

Existing tools of specifying NFPs tend to focus on specific properties, and also generic modelling languages cannot be used to specify NFPs of component-based system. Driving a comparison between theses tools and generic languages, we have identified that there is no such existing language that can be used to generically model NFPs; and it has formal semantics with support for CBSEs and less complex. Therefore, the objective is define such language that can be usable to define NFPs of component-based systems.

3
A Meta-Model for QML/CS

This chapter defines a quality modelling language (QML/CS), a specification language for modelling the non-functional properties (NFPs) of component-based software (CBS) through the creation of a new meta-model. This is achieved using the simplest technology available and that problems encountered will be discussed in subsequent chapters.

The structure of the chapter is as follows. Subsection 3.1, introduces and the contextualises the work. Subsection 3.2 provides details of the language architecture used, and in Subsection 3.3 QML/CS is defined. Subsections 3.3.2 to 3.3.7 provide a meta-model of relevant concepts, and a discussion concerning substitutability and conformance between constructs in QML/CS.

3.1 Introduction to QML/CS

Zschaler's [110] thesis defines the Semantic Framework for Non-functional Specifications of Component-Based Systems using measurements to specify the NFPs of component-based systems. It offers a framework defined based on extended temporal logic of actions (TLA+), deriving the NFPs of components and services from these measurements. TLA+ offers a logical foundation upon which to specify details and reason about concurrent systems.

Zschaler's framework provides five types: service, component, resource, container, and measurement, to represent its core concepts. These can be used to specify the different levels of abstraction required to specify the NFPs of component-based systems formally. The concepts of service and component complement one another according to the supposition that service is an interface provided to expose the functionality implemented in the component. The concept of a resource is defined so that the component and service can be associated with the required resource; enabling the requirement and the availability of the resource to be formally specified. The concept

of container is defined that can be used to give combined representation to other concepts like component, the service it provides and the resources it may require to perform a task. The concept of measurement is defined to specify NFPs linked with either a service or a component, and this is also used to mention constraints for NFPs.

Zschaler's framework conceptualised a context and application model to separate the abstraction of specification from its application to a specific application domain. The non-functional behaviour of a component or service is defined by a set of states that indicate the internal structure of the non-functional property being specified. The context model expresses a state machine model; this is independent of where the context model will be applied. This affords the liberty to specify measurements that are independent of the concrete application where it will be used; this also means that a context model might have more than one concrete application. The application model provides a concrete application of the context model; therefore, it is more closely associated with a real system wherein it is being applied to derive concepts from the context model to be applied to specify NFPs. Both the context model and application model are important, because they provide means to define measurements independently of the application, and to map the behaviour of measurements.

The QML/CS language is based on Zschaler's framework, but provides modified definitions of some concepts, such as the context model and application model. These modifications arise because the concepts of context and application model are ambiguous because the different parameters of measurements represented by same context model may exhibit different behaviour. This ambiguity affects the ability to define measurement based on the context model. The QML/CS language introduces a new class diagram that assists in mapping the context model to measurement definition and therefore provide clarity in terms of how measurements are formally defined, and such that their definition confirms to the context model. Consequently, the structure of the context model is modified to include a class diagram alongside the state machine model presented in the framework.

A summary of the achievements in this part of the thesis are as follows:

1. Language definition for QML/CS.
2. A comprehensive grammar for concrete syntax of QML/CS.

3.2 Language Architecture

This section explains the architecture of the QML/CS meta-model. It also shares an overview of the main packages, and shows how they are applied to organise QM-

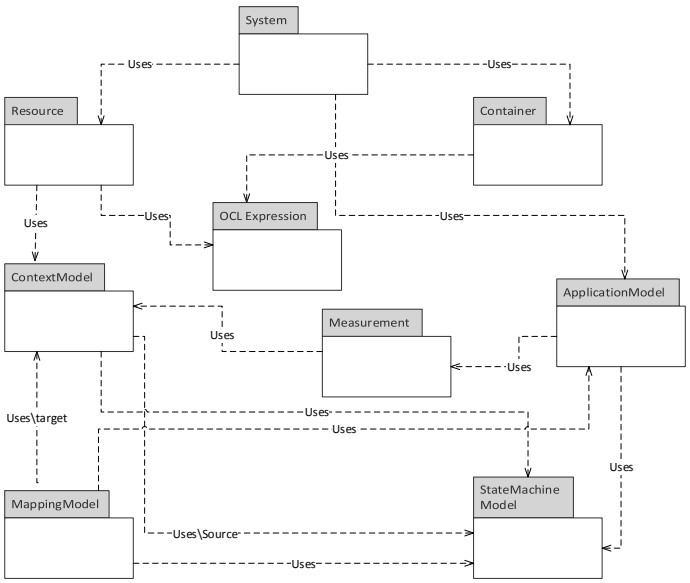

Figure 3.1: The high-level organisation of the QML/CS meta-model.

L/CS's high-level organisation. It describes the package structure of the QML/CS meta-model, and the related OCL meta-model. Moreover, it defines language specifications, models and reusable metalanguage OCL. Fig 3.1 provides information concerning each package and each class in the QML/CS meta-model. There is a package for each core concept so that these concepts can be reused in the context in which they are being applied.

The meta-model for QML/CS is composed of nine packages because there is a package for each core concept, as shown in Fig 3.1.

- Context Model Package:
  This package supports provision of an independent description of a specific application. It contains a number of concepts that are necessary to describe in abstract how the application works. It also links instances of the context model classes to the relevant state machine model, and is used by the measurements to define the behaviour of the measurement parameters. The model mapping package uses this package so that the behaviour of a measurement and its parameters can be validated based on the contextual use of measurement parameters in a specific application context. It also contains the class diagram concept introduced to remove ambiguity in the context model presented in Zschaler's framework.

- Application Model Package:
  The application model package represents a meta-model for defining applications. A specific application model can be seen as an instance of the context model designed for a specified use. The application model provides a basis for validating the context model, because it is used to reference and map entities in a context model to entities implemented in an application model. The application model can implement behaviour to manage measurements and their parameters in the context of their implementation, and provide an interface for mapping measurements to their abstract representations in the context model.

- Measurement Package:
  This contains the measurement concept based on the context model package, which makes it possible to describe a specific measurement. This package uses the context model package to define the behaviour associated with parameter types. This package can be applied to a specific operation by the application model.

- Mapping Model Package:
  This package is responsible for establishing the mapping between the measurement context model and the relevant application model, and is implemented to apply the context model to a specific scenario. It uses a number of mapping strategies to map the application model with its measurement context model: class mapping, state machine model mapping, state mapping and transition mapping. These mapping strategies can validate the behaviour of the measurement parameters in comparison with the behaviour of the operations to which the measurement is being applied.

- State Machine Model Package:
  This package is used by the context model package to define the behaviour of the measurement's parameters, and the behaviour of the operation in the application model. It allows the definition of the state machine model, states and transitions. The model mapping package can use this package to validate the operations in the application model against the measurement's parameters in the context model.

- Resource Specification Package:
  The resource package is necessary to define the context and environment being considered to define the measurement and its associated constraints. This package is used to link the resource specification with the context and application models, and is associated by the container package with the component or service being specified for the non-functional requirement.

- Container Specification Package:
  The primary purpose of the container package is to link the definitions of various entities within the system, such as defining the resource and linking it with concrete concepts and component and the service being provided by the specified components. The idea here is to organise the definition of these concepts in such a way that they appear to be linked to each other and can be evaluated as a combined definition of the container element. This is important because it defines relationships between those concepts and contributes to representation of the System.

- System Specification Package:
  The system package represents the the system's integrated perspective, in which we can locate all the partial definitions of the system. The package links them together to give a complete system specification. The system also represents instances of all packages and links those instances to provide an overall picture of the system's specifications.

- OCL Expression Package:
  This package defines a variant of OCL. It uses measurement, resource, container, and application model packages to specify OCL expression and identify any constraints on the behaviour of these design elements.

3.3 QML/CS Meta-Model

This section discusses the fundamental models and specifications of QML/CS and provides an overview of the QML/CS meta-model. The meta-model comprises the following parts excluding OCL meta-model concepts:

- Context Models and Application Models: Context Models are required to allow a measurement to be described independently of its concrete application, whereas application models are used to apply the context model to a concrete definition.

- Services: representing one of many services provided by a system.

- Component: representing a functional part of the system that provides a service.

- Measurement: representing the NFPs of systems and the constraints associated with them.

- Resource: representing the environment in which the application will run and the demand for NFP's specification.

- Container: combining the separate definitions of measurements, the component or service where the measurement is applied, and the required resources.
- System: it delivers overall picture of the system specification, which may have more than one container, and their internal definitions.

Throughout, we use OCL [83] as an expression language; To achieve this it was necessary to extend OCL in some places. These extensions, together with the core QML/CS meta-model, are described below as a subsection per concept.

3.3.1 Context Models and Application Models

The context model definition in [110], as mentioned earlier, is ambiguous because the different parameters of measurements represented by same context model may exhibit different behaviour. Therefore, we extended the context model with a new class diagram to facilitate the expression and mapping of measurements. The class diagram links the context model with the definition of measurement and provides a mapping structure so that the parameters for the measurement can be evaluated to identify behaviour. The class and attribute concepts from the class model for the newly introduced class are used by OCL [83] to provide expressions in terms of measurement definition. The context model is specific in that it mentions relevant elements for the measurement for which it is being defined, but it is generic with regard to specifying the component, operation or attribute, keeping the definitions abstract.

The application model offers a concrete representation of the context model indicating the context of the domain in which it is being applied, so that all abstract representations in the context model are linked with the concepts from the application. This means that same context model can be applied to different situations, and that the application model will reflect the specialisation of the context model for each situation. It also indicates that there could be more than one potential application models for each context model, so that the context models can be applied generically to define measurements and their practical implementations. The context model, as can be seen in Fig 3.2, adds a new class diagram are also reflected in the application model, so that mapping between the context model and the application model is consistent and can be evaluated. That indicates that the application model can be considered an instance of the context model. Therefore, mapping of a specific application model to the context model is essential for the system in which the measurement is being applied.

Fig 3.2 shows the components of meta-model that we developed for QML/CS. It represents how context and application models can be represented as a meta-class, and

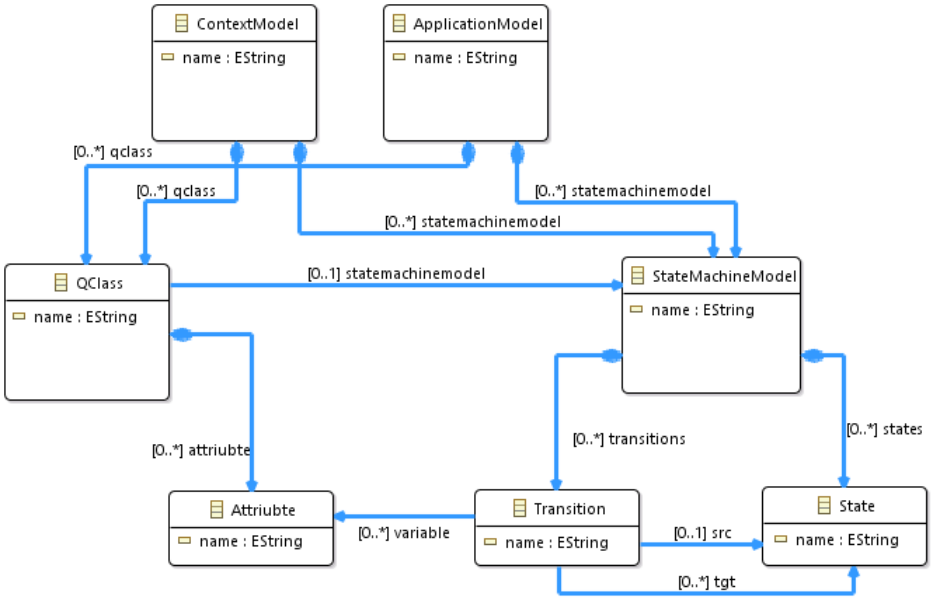

Figure 3.2: Context and Application Meta-Model

shows the core concepts of the meta-model for both models, as is expressed by state machine and the class-model of the newly added class diagram. The state-machine represents behaviour of the type of the parameter and therefore compatibility between types can be evaluated. Each context model can be used by a measurement to describe the behaviour of its parameters. It consists of the meta-class that is associated with its attributes and state machine model that contains a number of states and transitions. Transitions have source and target associations to link various states. Each application model can be used by a application to describe the behaviour of its operations. It is also linked to the meta-class that is associated with its attributes and in additional to a state machine model that defines target operations, which also consist of a number of states and transitions where each transition references source and target states.

Fig 3.3 shows an example of a context model defining the relevant steps in an operation call, in conjunction with a class model part named *ServiceOperation*. The transition indicates tasks that occur when the transition starts (as shown in Fig 3.3). If a service has not received any request to execute an operation, then the service should remain in the 'Idle' state. If a component has received a request, then the service goes to the 'RequestAvailable' position. Subsequently, when the service starts the process of handing the request, the service should move to the 'HandlingRequest' state. Once the process of handling the request completes, the service should return to either the 'Idle' or 'RequestAvailable' position, depending on whether a new request has been

received or not. The *ServiceOperation* class is defined in the context model to support the definition of a measurement.

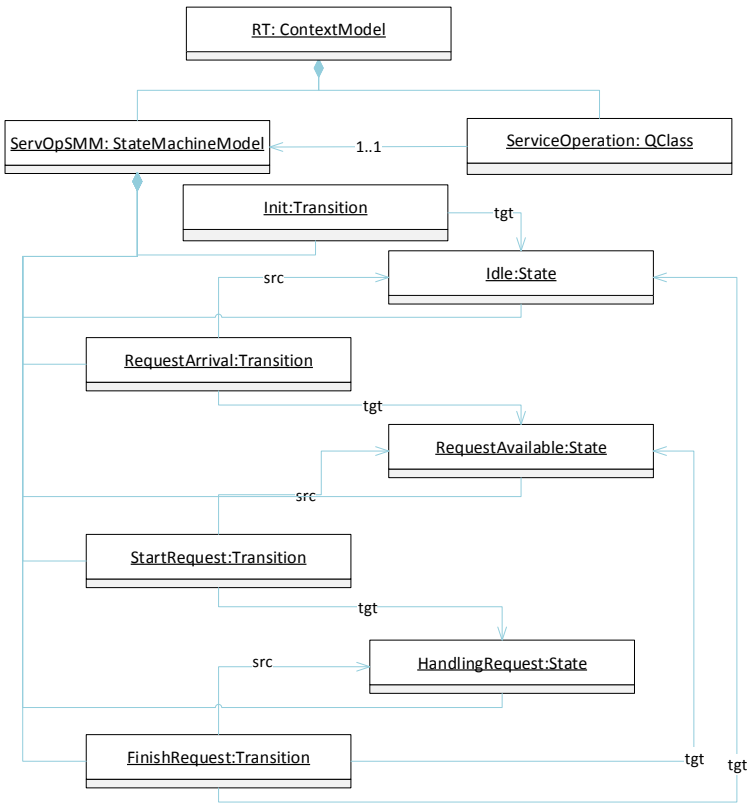

Figure 3.3: Example of Context Model.

The Fig 3.4 demonstrates an example of an application model for a component *Counter*, which implements two methods to increase the value of a counter variable and return the currently allocated value. It shows the different states encountered when performing an increment operation like ReceivedIncrement, StartingIncrement and FinishedIncrement to obtain the current value of the *Counter* variable: i.e. ReceivedGetValue, StartingGetValue and FinishedGetValue. The name and number of states for a component will differ according to on the type of feature being implemented, but this simple example highlights how an application model can be represented as a context model with specific states and their operations [110]. Another example of an application model could be for a component *Tracer* to assist in maintaining diagnostic information about a system; thus, when a request returns

trace information then the system is working. This implies the typical states of ReceivedLog, StartingLog, SavingLog, FinishedSavingLog and FinishedLog for the log operation, and ReceivedGetTrace, StartingGetTrace, ReadingLog, TransformingLog and FinishedGetTrace as a get trace method. Both these application models provide instances of the same context model that shows an abstract component, specifically implemented by an application model.

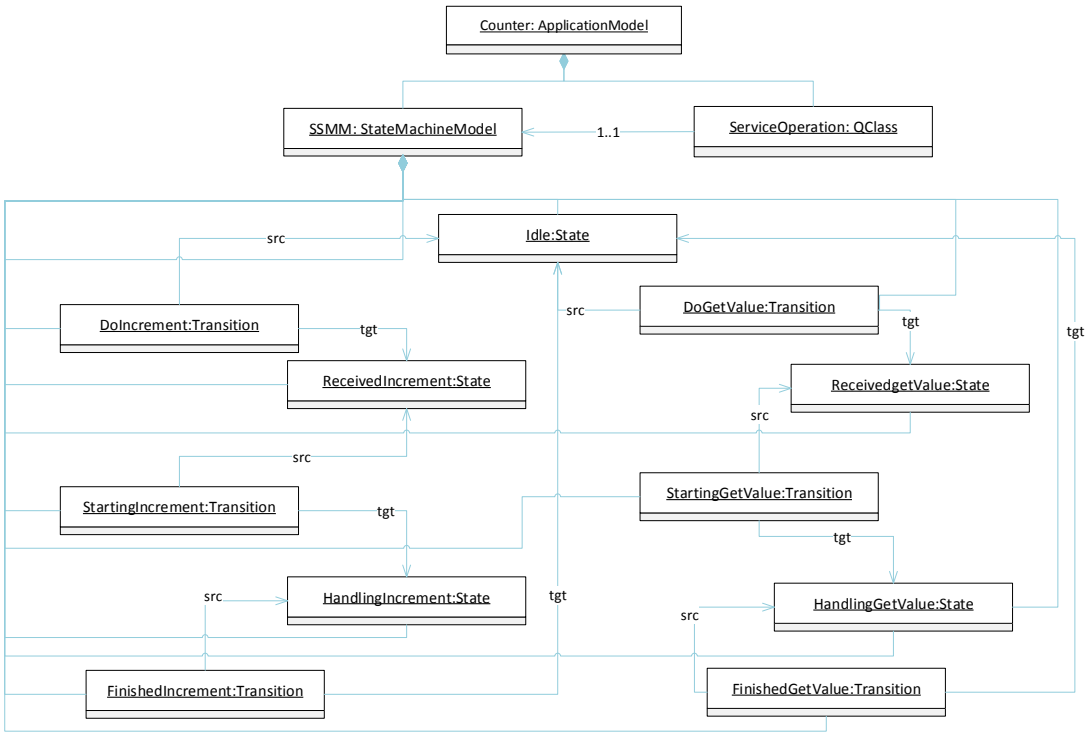

Figure 3.4: Example of Application Model.

3.3.2 Measurement

In line with [110], we consider NFPs as constraints over non-functional characteristics. These characteristics are measurements; that is, they map a given state system (and possibly a history of states) to the value of the property. Constructing a meta-model for measurement requires definition of the rules associated with main concepts.

The key elements are represented in the measurement meta-model, which is shown in Fig 3.5. A *MeasurementDeclaration* has properties *name* and *Data Type*. It references

InContextModelStatement so that a link to the measurement context model can be established. A measurement can have one or more parameters, which is represented by a link to *MeasurementParam*. *MeasurementParam* has a property *name* and a reference to *QClass* type. In addition, *MeasurementDeclaration* includes a measurement context named *MeasurementCxt* that gives a context for which a measurement can be specified along with *ConstraintCS* for *specConstraints*, which is a part of OCL expressions.

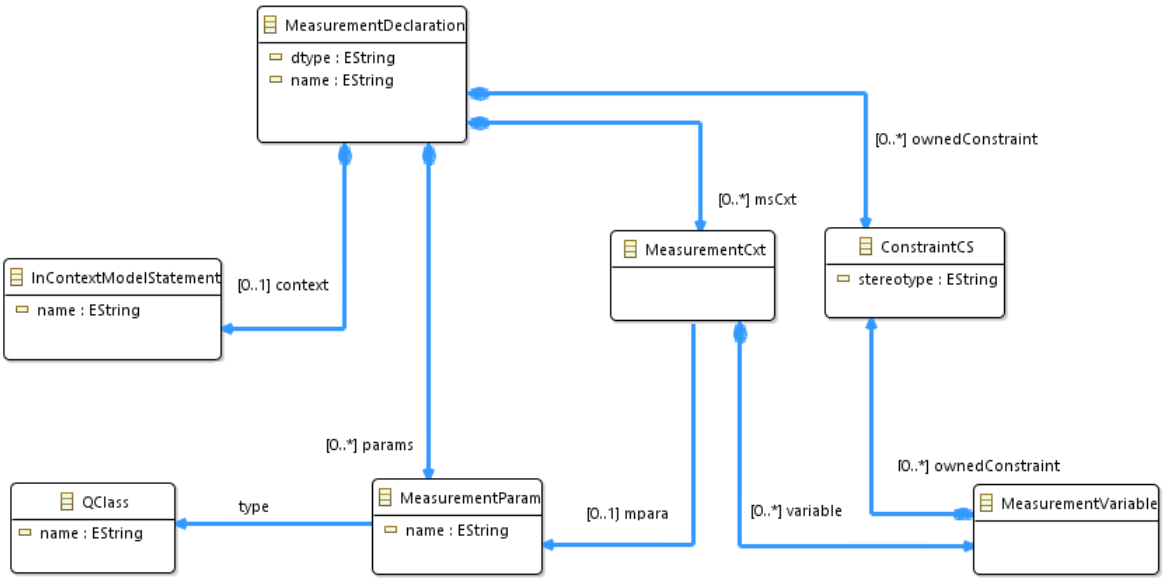

Figure 3.5: Measurement Meta-Model.

Listing 3.1 shows QML/CS measurement declaration syntax. The rule *MeasurementDeclaration* starts with the feature named *context* to reference the rule *InContextModelStatement* that refers to the reference of measurement context model (as specified in lines 9 and 10). Line 3 begins with keywords 'declare' and 'measurement', followed by *dtype* and *name* features which represent a data type and name for the measurement. Line 4 mentions that a *MeasurementDeclaration* contains an arbitrary number (*) of *MeasurementParam* which are added (+=) to a feature called *params*. Line 6 also shows that a *MeasurementDeclaration* contains an arbitrary number (*) of *MeasurementCxt* which are added (+=) to a feature named *msCxt*. The rule *MeasurementCxt* starts with a keyword 'on', followed by feature *mpara* which represents a parameter name. The feature *trans* references a transition that comes from a measurement context model. It demonstrates what happens in each transition of context model for this measurement specification indicating change in variables associated with transition as can be seen in lines 13 thru 15. The rule *measurement parameter* that has a type

```
1  MeasurementDeclaration:
2      context=InContextModelStatement
3    'declare' 'measurement' dtype=Type name=MeasurementID '('
4      (params+=MeasurementParam (',' params+=MeasurementParam)*
5      )?')' '{'
6      (msCxt+=MeasurementCxt(msCxt+=MeasurementCxt)*)?
7        (ownedConstraint+=specConstraints)*
8          '}';
9  InContextModelStatement:
10   'in' 'context' name=ImportName';';
11 MeasurementCxt:
12   'On' mpara=[MeasurementParam|MeasurementArgumentId]'.'
13   (trans=[qmlcsmm::Transition|UnrestrictedName]
14     (variable+=MeasurementVariable
15       (variable+=MeasurementVariable)
16     ?)
17   ?);
18 MeasurementParam:
19   {(type=[qmlcsmm::QClass|UnrestrictedName] name=MeasurementArgumentId);
20 specConstraints returns ConstraintCS:
21   stereotype='spec' (specification=SpecificationCS ';') ;
```

Listing 3.1: Measurement Syntax in QML/CS.

of *QClass* is represented in lines 18 and 19.

An example of measurement *response time* as specified via QML/CS is shown in listing 3.2. This illustrates how one measurement, called *responseTime*, is defined using QML/CS measurement syntax. It is essential that a context model exists to allow measurement specification be expressed. In other words, that measurement uses a context model when defining *response time*. Once a measurement definition is specified, a designer can express the NFPs of the application as constraining predefined measurements.

3.3.3 Component and Service

As mentioned previously, a component represents a functional part of the system, providing a service as one of many interfaces provided by the system. Because these operations are among the components and measurements called by the components or the services being provided by the components, this provides differentiation between NFPs attached to components or services. The NFPs attached to a component operation typically reflect the inner state of the operation, such as execution time, and the NFPs attached with service operations that reflect consideration of the service operation as a blackbox and therefore demonstrating the external perspec-

```
1  in context RT;
2  declare measurement Real response_time (ServiceOperation op){
3  On op.Init update
4          ResponseTime = 0;
5          hadOpCall = FALSE;
6  On op.RequestArrival update
7                  start = 0;
8                  end = 0;
9  On op.StartRequest update
10                 start = now;
11 On op.FinishRequest update
12                 end = now;
13                 ResponseTime = ResponseTime + end;
14                 ResponseTime = ResponseTime - start;
15                 hadOpCall = TRUE;
16 }
```

Listing 3.2: *Response time* Measurement specified via QML/CS.

tive of the service operation, such as *response time*. This differentiation results in the categorisation of measurements into intrinsic and extrinsic measurements, where intrinsic measurements represent component operation related measurements, and extrinsic measurements represent service operation related measurements. This will be reflected by the types of measurement parameters, as one of the parameters of measurement is the operation for which the measurement is being specified. Type of the operation parameter will give an insight into whether it is a service operation or a component operation.

The QML/CS relies on utilising the concepts of component and service to specify the NFPs so that both intrinsic and extrinsic measurements can be identified based on specification. They are also mentioned in the meta-model, to indicate the differentiation in type of parameters for measurement, so that both the context model and the application model can invoke relevant operational parameters to measure the non-functional property. The class *MeasurementCall* was defined to extend OCL, to facilitate invocation of the measurement, also considering component and service types, so that mapping of the measurement call can be linked with associated operations. Fig 3.6 shows how the component and service are represented in QML/CS.

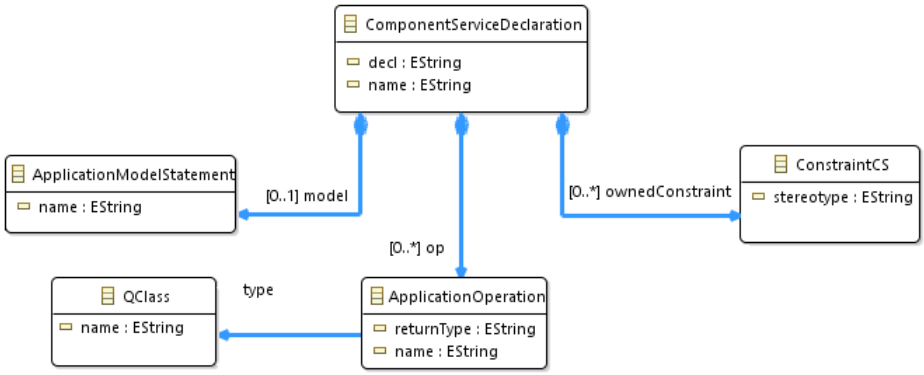

Figure 3.6: Component/Service Meta-Model.

Listing 3.4 shows QML/CS component service declaration syntax. The rule *ComponentServiceDeclaration* starts with the feature named *model* to reference the rule *ApplicationModelStatement* that refers to the reference of application model (as specified in lines 1 and 2). Line 6 begins with keywords 'declare' and 'Component|Service', followed by *name* feature which represents name for the component or service. Line 7 starts with keywords 'uses|provides' and then mentions that a *ComponentServiceDeclaration* contains an arbitrary number (*) of *ApplicationOperation* which are added (+=) to a feature called *op*. Line 8 shows that a *ComponentServiceDeclaration* contains also an arbitrary number (*) of *alwaysConstraint* which are added (+=) to a feature named *ownedConstraint*. This section might use OCL to specify the expressions required to define the measures and constraints on the defined non-functional property. For example if we wish to specify the response time of an operation as not more than sixty seconds, OCL expression will be beneficial in defining that consistently.

The rule *ApplicationOperation* starts with a feature named *type* that represents the type of operation, followed by features *returnType* which represents a returned data type, and *name* that indicates the name of operation. The operation signature includes the *type* and helps when evaluating the measurement parameters, so that the behaviour of the operation can be validated. An operation is a functional element of the component for which a non-functional property has to be measured; and functionality is the service provided by the component. The syntax for writing the operation signature is also included, so that the return type, name and parameters of the operation are clearly specified for each operation.

For example, listing 3.4 shows an example of the application model specified in QML/CS. It demonstrates a service named *Counter*, which has *getData()* operation. The specification of this application places constraints over measurement *response time*

```
ApplicationModelStatement:
  'application' name=ImportName ';';

ComponentServiceDeclaration:
  model=ApplicationModelStatement
  'declare' decl=('Component'|'Service') name=ComponentOrServiceNameID
    '{'
  (('uses'|'provides') (op+=ApplicationOperation) '()' ';')+
   (ownedConstraint+=alwaysConstraint)*
  '}';

ApplicationOperation:
  type=[qmlcsmm::QClass|UnrestrictedName] returnType=Type
  name=ApplicationOperationId;

alwaysConstraint returns ConstraintCS:
  stereotype='always'(specification=SpecificationCS ';') ;
```

Listing 3.3: QML/CS Application Declaration Syntax

of *getData()*, to be always less than 60 milliseconds. This includes a reference to a model mapping *GD2RTMapping* in order to validate the behaviour of this operation confirming to the behaviour of *response time* parameter.

```
application CounterModel;
declare Service Counter {
  provides Operation int getData();

  always response_time (getData by GD2RTMapping.Mapping1)<60;
}
```

Listing 3.4: Counter application specified via QML/CS

3.3.4 Resource

A component-based system demands a hardware machine to run on, and it can perform its function only if the required resources are available. This makes the existence and availability of resources important for the application and some NFPs, such as response time and execution time depend on the available resources. The same NFPs may have different expectations based on the kinds of resources available and the constraints applied when using them. On a granular level, each task that the system must perform would require resources like CPU, memory and the availability of required processing or storage in the memory, which would be critical for task completion. In line with [110], we consider that in a system with a resource such as a CPU, capacity limit of the resource can play important role in deriving the NFPs of

a system. There is no resource with unlimited capacity, and not all resource capacity can be allocated to a single application as typically applications share resources with other applications running on the system, or at least with the operating system. A well specified non-functional property should consider the constraints on resource availability and usage so that the behaviour of the system can be predicted in different working environments. Resources can be direct or indirect from an application prospective, with direct resources, like CPU, being specifically requested by the application, and resources like power, that is needed by the CPU and the application, not being specifically requested. Similarly other resources like memory have same implications when specified. Therefore, it should be considered when specifying NFPs that only resources that are specified can be directly requested by the application and therefore be measured.

In [110] Zschaler provides three layers for the specification of resource. First, a resource-service layer, which is an abstract layer that defines the resource's service and provides information for the other two layers. Second, a resource-measurement layer, which provides a definition for the non-functional aspects of the resource through the use of history variables. Third, a resource-property layer, which provides a definitions of constraints over the second layer. In QML/CS language, we follow the same definition of resources as that provided in [110]. However, the definition then only considers a resource model with a state machine, whereas, in our language we extended the definition of a resource with a meta-class and a state machine, as can be seen in Fig 3.7. Constructing a meta-model for a resource requires definition of rules informing designations of main concepts that are abstract and concrete, as shown in Fig 3.7. We first defined an abstract resource, so that the interface could be used to define resource demands and check the resource's capacity. We then specified a concrete resource by specialising an abstract resource and a concrete *capacitylimit* in addition to with constraints over the capacity of the resource.

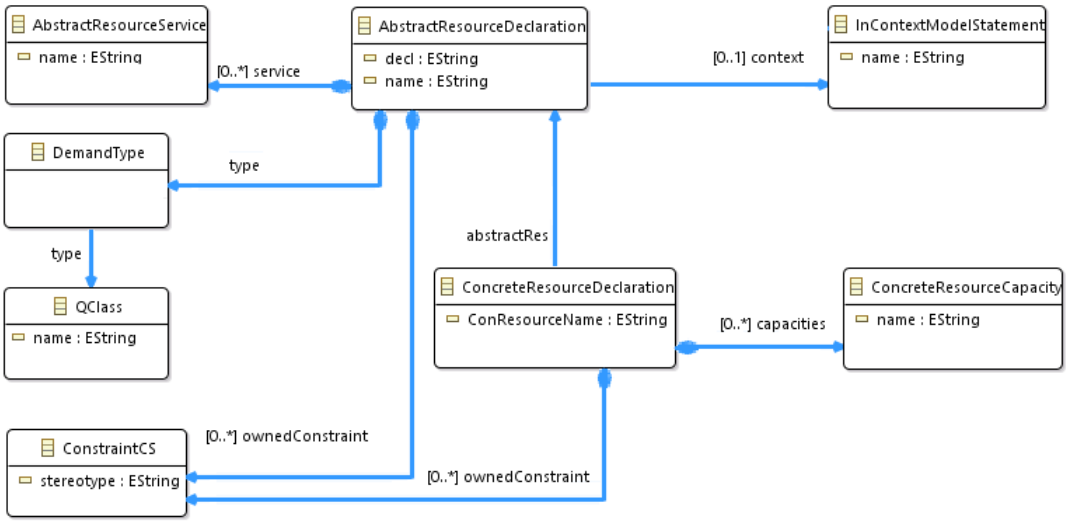

Figure 3.7: Resource Meta-Model.

Listing 3.5 shows QML/CS abstract resource declaration syntax. The rule *AbstractResourceDeclaration* starts with the feature named *context* to reference the rule *InContextModelStatement* that refers to the reference of resource context model. Line 3 begins with keywords 'declare', 'abstract' and 'resource', followed by *name* feature which represents a name for the abstract resource. In addition, line 4 starts the keyword 'demand', followed by *type* feature that shows the type of expression that the individual resource demands. Line 5 mentions that *AbstractResourceDeclaration* contains *AbstractResourceService* which is added (+=) to a feature called *service* to express the service provided by the resource. Line 6 shows that *AbstractResourceDeclaration* contains *alwaysConstraint* which is added (+=) to a feature named *ownedConstraint* to indicates that capacity limiting conditions should be satisfied, to enable the resource to provide its service.

For example, listing 3.6 demonstrates an example of the abstract resource named *CPU*. The abstract resource references a context model named a *CPUModel*, which has a *state-machine model* that continuously assigns tasks to the resource and a *Task* class that demands this type of resource. The *CPUModel* also contains a function called *timeAlloted* to allocate the time taken by *CPU* to perform each task. *Task* execution on the *CPU* is achieved by collection *scheduledTasks* of *id* and *demand* tuples. *Task* indicates the specification needed to define an individual task demand, which includes attributes such as *period*, *worst-case execution time* and *deadline*.

```
1 AbstractResourceDeclaration:
2   context=InContextModelStatement
```

```
3   'declare' 'abstract' decl='resource' name=AbstractresourceNameID '{'
4   'demand' type=DemandType ';'
5   (service+=AbstractResourceService)+
6   (ownedConstraint+=alwaysConstraint)+
7 '};'
8 DemandType:
9   type=[qmlcsmm::QClass|UnrestrictedName];
10 AbstractResourceService:
11   'service' '(' ownedType=CollectionTypeCS name=ID ')' '=';
12 alwaysConstraint returns ConstraintCS:
13   stereotype='always'(specification=SpecificationCS ';') ;
14 InContextModelStatement:
15   'in' 'context' name=ImportName';';
```

Listing 3.5: QML/CS Abstract Resource Syntax Declaration

```
1  in context CPUModel;
2  declare abstract resource CPU {
3    demand Task;
4
5    service (Set (Task) demand) =
6    always (scheduledTask->collect (t | t.demand)->inlcudesAll(demand)and
       scheduledTask->size()= demand->size() and scheduledTask->forAll (t |
       timeAlloted (t.id)>= t.demand.wcet));
7
8    always (capacityLimit (demand)
9    => service (demand));
10 }
```

Listing 3.6: CPU resource specified in QML/CS

In the second step, a concrete resource is specified as detailed in listing 3.7. Listing 3.7 shows QML/CS concrete resource declaration syntax. The rule *ConcreteResourceDeclaration* starts with keywords 'declare', 'resource' and 'resource', followed by *ConResourceName* feature which represents a name for the concrete resource. It is for the specialising an abstract resource *AbstractName* via the feature named *abstractRes*. Line 3 mentions that *ConcreteResourceDeclaration* contains *ConcreteResourceCapacity* which is added (+=) to a feature called *capacities* to specify a concrete capacity limit for the resource. Line 4 also shows that *ConcreteResourceDeclaration* contains *Constraints* which is added (+=) to a feature named *ownedConstraint*, which is used to place conditions over the capacity of a resource.

The example of a concrete resource*CUP*, with a scheduler based on rate-monotonic scheduling(RMS)[1], as shown in listing 3.8 explains the process of defining the *CPU*

[1]Scheduling the time allocated to periodic hard-deadline real-time users of a resource. The users

capacity limit. OCL expression is used to place constraints on the standard RMS schedulability criterion, as part of the capacity limit for CPU specification.

```
ConcreteResourceDeclaration:
  'declare' 'resource' ConResourceName=ID 'of' abstractRes=[
   AbstractResourceDeclaration|AbstractresourceNameID] '{'
   (capacities+=ConcreteResourceCapacity)+
    (ownedConstraint+=Constraints)+
   '}';
ConcreteResourceCapacity:
  'capacityLimit' '(' ownedType=CollectionTypeCS name=ID ')' '=';
Constraints returns ConstraintCS:
   (specification=SpecificationCS ';') ;
```

Listing 3.7: QML/CS Concrete Resource Syntax Declaration

```
declare resource RmsCpu of CPU{
  capacityLimit (Set (Task) demand))
     = demand->iterate (t: Task; acc: Real | acc + t.wcet /t.deadline)
     <= demand->size() * (2.sqrt(demand->size()-1);
}
```

Listing 3.8: RMS Scheduled concrete resource specified in QML/CS

3.3.5 Container

Linking available resources and components requires a formal method for specifying how the those two concepts are used in a system. Zschaler [110] introduced a concept called container, which represents a relationship between resources and components and defines a container strategy. This container strategy provides information about intrinsic properties, required resources, and a description of the extrinsic properties a given container can provide, taking into account the intrinsic properties and required resources.

The container is responsible for integrating the service, the component providing the service and the resources needed by the components, so that their association is clear. Therefore, the container is a place in which all these links are specified for use for reference. In accordance with [110], QML/CS uses the *requires* keyword to state the container input strategy (intrinsic properties and required resource by a container) and *provides* keyword to specify the service provided by the container. A meta-model

are assigned priorities such that a shorter fixed period between deadlines is associated with a higher priority. RMS provides a low-overhead, reasonably resource-efficient means of guaranteeing that all users will meet their deadlines provided that certain analytical equations are satisfied during the system design.

for the container is shown in Fig 3.8, which provides a definition of the association and the relationship between the component, the service and the resources.

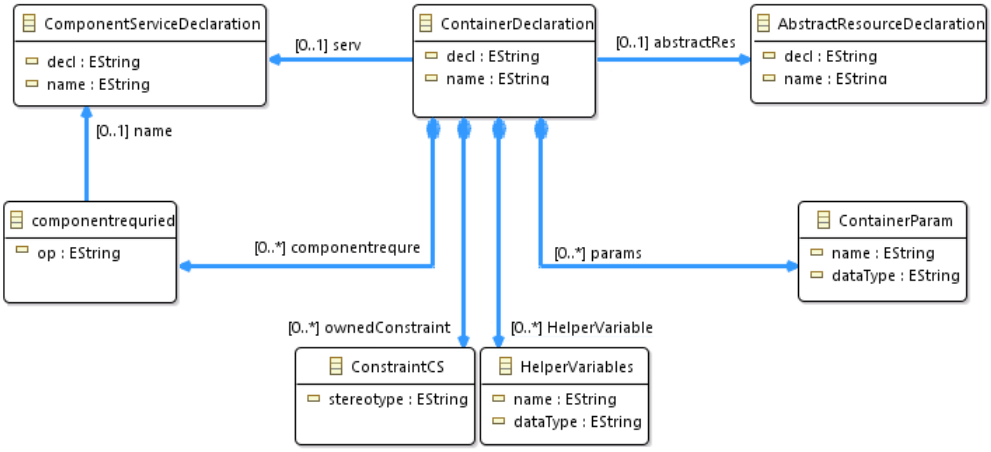

Figure 3.8: Container Meta-Model.

Listing 3.1 shows QML/CS container declaration syntax. The rule *ContainerDeclaration* starts with keywords 'declare' and 'container', followed by *name* features which represent a name for the container. Line 2 mentions that a *ContainerDeclaration* contains an arbitrary number (*) of *ContainerParam* which are added (+=) to a feature called *params*. Line 3 also shows that a *ContainerDeclaration* contains an arbitrary number (*) of *HelperVariables* which are added (+=) to a feature named *HelperVariable*. The optional container parameters are be passed to the container and any extra information required to measure or compare the NFPs respectively. Line 4 starts with keywords 'requires' 'component', followed by the rule *componentrequried* that references to allows the declaration of a component so that *ComponentPattern* specification can be accessed vai the feature *componentrequre*. Lines 5 starts with keyword 'resource', followed by the feature *abstractRes* to include the resource specification in the container that has some constrains, by which a capacity requirement for some abstract resource is specified. Lines 8 thru 13 show the specification to express that there will be a top-level component offering an interface for service implementation, because the service can the be executed by more than one component. Then, the *implemented by* keywords the component can be referenced by a name. For example, listing 3.10 demonstrates an example of a container named *SimplerContainer*, based on the specifications from the listing 3.9.

3.3.6 System

The system provides information about the instance of each concept that is combined together with the container used to link them so that the overall representation of

```
1  ContainerDeclaration:
2    'declare' decl='container' name=ContainerNameID '('(params+=
     ContainerParam (',' params+=ContainerParam)*)? ')' '{'
3      (HelperVariable+=HelperVariables (','HelperVariable+=HelperVariables
     )*)?
4     (('requires' 'component') (componentrequre+=componentrequried(','
     componentrequre+=componentrequried)*))*
5      'resource' abstractRes=[AbstractResourceDeclaration|
     AbstractresourceNameID] '('
6        ('((ownedConstraint+=SpecificationCS) (',' ownedConstraint+=
     SpecificationCS)* )?')'
7        ')'';'
8      'provides' 'service' 'implemented by' serv=[
     ComponentServiceDeclaration|ComponentOrServiceNameID]'{'
9          (ownedConstraint+=(Constraints|alwaysConstraint))*
10   '}'
11 '}';
12 HelperVariables:
13   name=HelperVariablesID ':'dataType=Type ';';
14 ContainerParam:
15   name=ContainerParamID ':'dataType=Type;
16 alwaysConstraint returns ConstraintCS:
17   stereotype='always'(specification=SpecificationCS ';') ;
```

Listing 3.9: QML/CS Container Syntax Declaration.

```
1  declare container SimpleContainer
2  (ResponseTime: Real) {
3  ExecutionTime: Real;
4  requires
5    component C {
6    provides op1();
7    always execution_time (op1) <
8    ExecutionTime;
9    };
10
11   resource CPU.canHandle (
12   Set{Task(
13   period = ResponseTime,
14   deadline = ResponseTime,
15   wcet = ExecutionTime)
16   });
17
18 provides service implemented by C {
19   ExecutionTime < ResponseTime =>
20   always response_time (op1) < ResponseTime
21   }
22 }
```

Listing 3.10: Example: Simpler Container syntax.

the system is clear. All the above meta-classes should be composed to obtain a global view of the system. A system specification binds together component, resource and container specifications. As stated in [110], it links the intrinsic specifications of components, the resource specifications of available resources with a container strategy connecting both the components and the resources in a system. In QML/CS language, we use the definition of system specification and construct a meta-model for a system that requires definition of the rules between its main concepts as shown in the Fig 3.9. Listing 3.11 shows the QML/CS syntax for how a system is specified. The main elements should be represented in the meta-model with rules as follows: a system has a *Name*, *instanceList*, reference of *systemContainer*, *ComponentsAndResources* and *serviceProvided*. These keywords are replaced by concrete representation when writing the specification. A system and one *container use* defined components and resources to link resources and components to provide a service. This is achieved by placing an instance on the parameters of the container using *uses* key words and *ComponentsAndResources* that indicates a comma-separated list of instance names as specified in *InstancesList*.

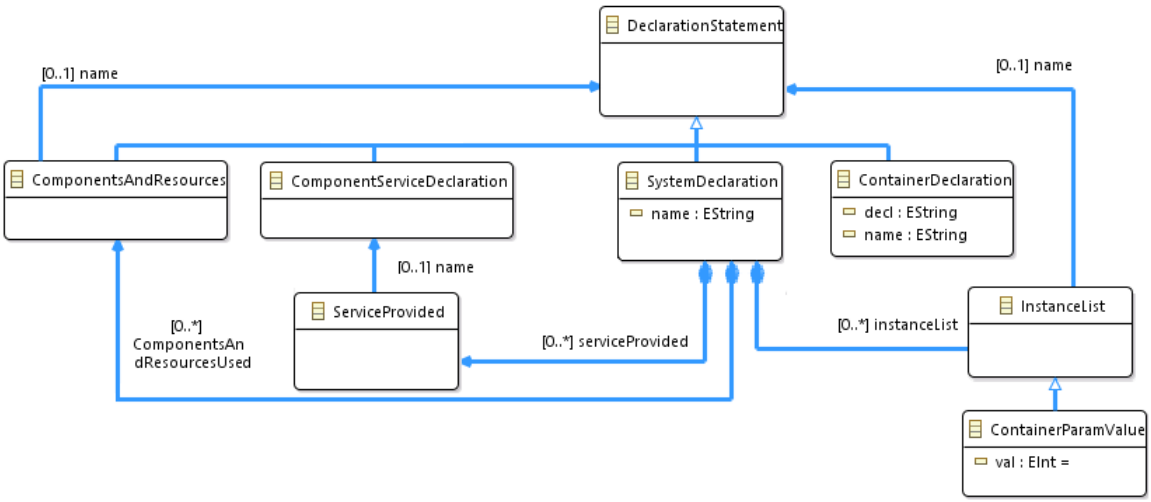

Figure 3.9: System Meta-Model.

Listing 3.12 shows an example of a complete system specification, which has three instances: *CounterComp c*, *RmsCpu cpu* and *SimpleContainer(60) container*. The container uses component called *c* and the resource called *cpu* to provide Counter *cServ*.

```
DeclarationStatement:
MeasurementDeclaration | ComponentServiceDeclaration
| AbstractResourceDeclaration | ConcreteResourceDeclaration
|ContainerDeclaration|SystemDeclaration
;
SystemDeclaration:
'declare' 'system' name=ID '{'
  ('instance' (instanceList+=InstanceList) ID ';')*
  ('container'
     'uses' (ComponentsAndResourcesUsed+=ComponentsAndResources(','
    ComponentsAndResourcesUsed+=ComponentsAndResources)*) ';')*
      'container'
      'provides' (serviceProvided+=ServiceProvided';')*
  '}'
;
InstanceList:
name=[ComponentServiceDeclaration|ComponentOrServiceNameID]| name=[
    AbstractResourceDeclaration|AbstractresourceNameID]|
    ContainerParamValue
;
ComponentsAndResources:
 name=([ComponentServiceDeclaration|ComponentOrServiceNameID] | [
    AbstractResourceDeclaration|AbstractresourceNameID])
;
ServiceProvided:
  name=[ComponentServiceDeclaration|ComponentOrServiceNameID]
;
ContainerParamValue:
name=[ContainerDeclaration|ContainerNameID] '('val=NUMBER_LITERAL')'
;
```

Listing 3.11: QML/CS System Syntax Declaration.

```
declare system CompleteSystem {
instance CounterComp c;
instance RmsCpu cpu;
instance SimpleContainer(60) container;

container uses c, cpu;
container provides Counter cServ;

}
```

Listing 3.12: System Specified in QML/CS.

3.3.7 Extension of the OCL Meta-Model: QML/CS

There is a need for an expression language to improve the expressiveness of the structural constraints of the modelling languages like QML/CS. There are different alternatives for expression languages like building our own expression language or reusing The Object Constraint Language (OCL) [83]. Building our own expression language involves doing a great deal of work to support a part of QML/CS specification. Also writing a complete new language when a well tested expression language exists and also supports extension is not justified. Using OCL and extending it where needed allows to focus on providing a specification language rather than working on writing an expression language solely to support this new specification language. OCL is commonly used as an expression and constraint language in MDE to support modelling languages with constraints [59].

In our work, we were more concerned with the practical side of expression language and its ability to provide support for QML/CS specification. OCL is very helpful in specifying expressiveness in a modelling language. It is originally defined as closely linked to UML; however, in this thesis we used OCL meta-model independently. Fig 3.10 shows the OCL meta-model, which expresses a variety of types of expression that can be used in QML/CS language. There is a need to extend OCL expression to enable the user of QML/CS language to specify QML/CS expression invariants. However, the current OCL standard does no support behaviour invariants that we want to use for call expressions in QML/CS. A number of approaches like [24] and [108] that extend the OCL to enhance the expressiveness of behavioural constraint. These approaches include a number of temporal logic operators to enhance the formal specification capabilities of OCL. It is important that the extension of OCL can be integrated easily with the structure of QML/CS so that QML/CS can define its expressions based on extended OCL without requiring any specially developed components. The required simple OCL extension can be achieved by creating a specific OCL CallExpression to extend the existing OCL standard expression package. For each QML/CS expression we define derived call expressions, such as *measurementCallExp*, *capacityLimitCallExp* and *resourceServiceCallExp* to extend the capability of OCL standard expression *FeatureCallExp* and provide requisite invariants for these QML/CS components. Fig 3.11 shows these extended OCL expressions.

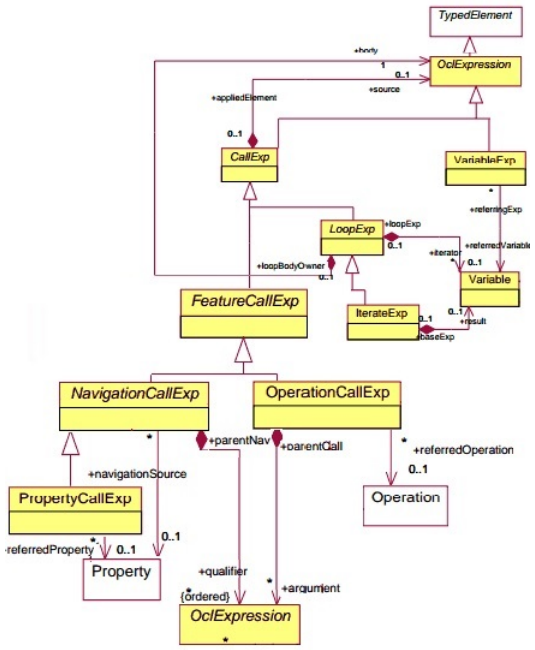

Figure 3.10: OCL meta-model derived from [83]

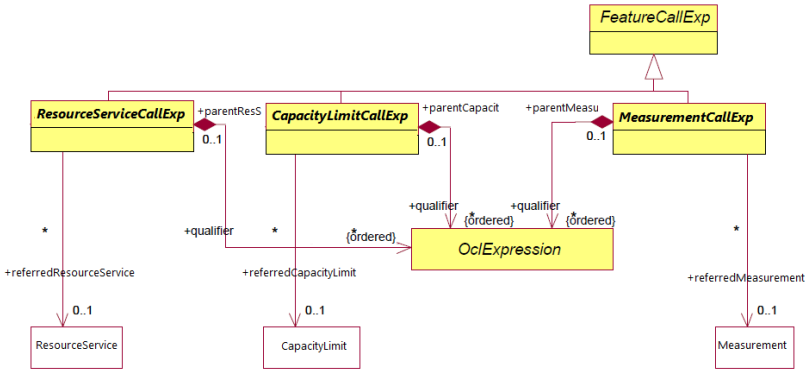

Figure 3.11: Extended OCLCallExpression

3.4 Summary

The overall aim of this chapter is to illustrate the process of defining QML/CS language based on MOF standards. This chapter has presented language architecture, definition and core concepts of QML/CS language. Then, it shows the definition of

a meta-classes (meta-model for QML/CS) using MOF standard.

The solution presented in this chapter is not complete, because it does not show the limitations of using using MOF standard to define QML/CS language and the problem of applying a measurement to a concrete application. The results of this stage shows only definitions of models and specifications of QML/CS using standards language. These issues of using UML standards are addressed in the second stage of the research in chapters 4 and 5.

The next two chapters will address these problems by applying deep meta-modelling approach 'Clabject' to QML/CS and specifying mappings between context and application models, that is used to provide a complete definition for QML/CS language as discussed in chapter 6.

4

Context Model Definition Ambiguity

This chapter presents the first key challenge encountered when building a meta-model for QML/CS, which is the ambiguity of the context model definition. This challenge is addressed by applying the deep meta-modelling technique, which uses the arbitrary number of levels for modelling to remove the ambiguity of the concepts and link them in a clear and precise manner. Contrary to approaches like MOF, the deep meta-modelling exposes more than two levels to be used with the provision of linguistic and ontological concepts that help in giving different interpretation of the concepts based on the context of use.

4.1 Meta-Modelling

Approaches to meta-modelling are mainly classical four layered and deep. We have already discussed standard meta-modelling in Section 2.3.4, for example UML; however, it exposes only two levels of modelling to the user. This limits the user from specifying the system concepts that need more than two levels of modelling and therefore the ambiguous features of stereotypes are used to elaborate. There are many systems in which two level modelling is insufficient because the hierarchy of classes and their instances are not discrete and each instance has further classifications that require modelling. We can take the example of library system, such as the classification 'book', where each sub class of 'book' has further sub classes that have their own instance hierarchy, as can be seen in Fig 4.1. Let us consider a modelling system that could manage a library about 'book' item with emphasis on Java 8. The class 'book' appears at the top level as the meta-class, and the concept 'Java 8' is considered an instance of a book. The concept 'Java ' is itself a class because there are many further instances of 'Java 8', like *Java 8 lambda*, and so on. These instances are further classes, providing more instances of these concepts. Such a situation demands an unambiguous class and instance hierarchy, so that each level can exist as an instance of the upper level, but can work as a class in further instances down the layers; this can not be done in UML without introducing specific ambiguity about the concepts.

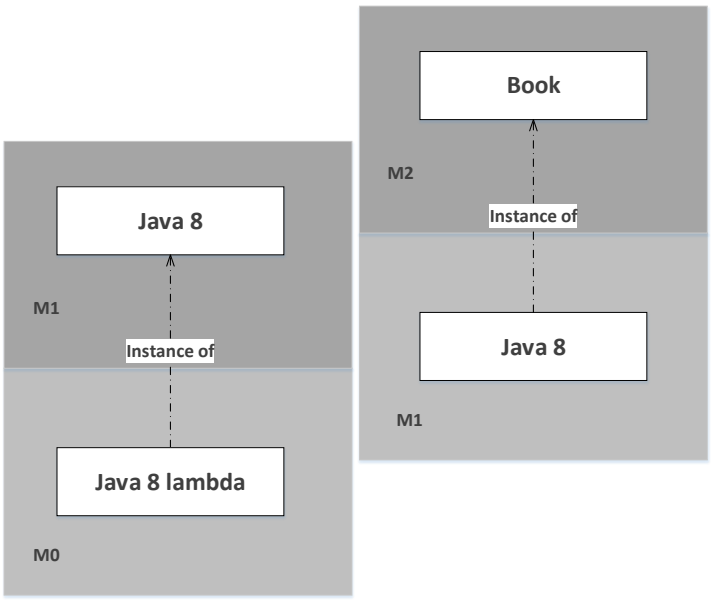

Figure 4.1: Example of Library Meta-modelling Hierarchy.

4.2 Ambiguity of Context Model Definition

Previous to our discussion in Subsection 3.3.3, listing 4.1 presents a concrete example of the application model. The purpose of this listing is to highlight that the operation *getData* is a concrete argument to the measurement *response_time*. The operation *getData* is an instance of *Operation* and the type of parameter expected by the measurement *response_time* is *ServiceOperation*. An initial attempt to model this example is given in Fig 4.2. The left part, (labelled (a)) depicts the *ServiceOperation*, which is defined as an instance of a *Class*, while the right part (labelled (b)) shows *getData*, which is defined as an instance of the *Operation* in the meta-level model.

```
1 application CounterModel;
2 declare Service Counter {
3   provides Operation int getData();
4
5   always response_time (getData by GD2RTMapping.Mapping1)<60;
6 }
```

Listing 4.1: Example: Counter application model specified in QML/CS

The parameters of measurement *response_time* requires a type to be *ServiceOperation* indicating that *getData* and *ServiceOperation* should have an instance-of relationship as can be seen in Fig 4.3. But *Operation* and *Class* are in the same level (M1), and *ServiceOperation* and *getData* are also in the same level (M0). An entity that exists as a type and instance at the same time like *ServiceOperation* can not be represented using the standard UML modelling because it is limited to two level of representation like a class and its instance. Therefore, *ServiceOperation* should move one level up so that the logical relationship between *getData* and *ServiceOperation* exist. It is important to ensure that these types are on their right levels of abstraction in the meta-model so that their association can be presented. The abstraction levels in a meta-model help in identifying the links between concepts of the meta-model at each level so that instance-Of relationship exist. The solution of this initial problem is presented in Section 4.3.

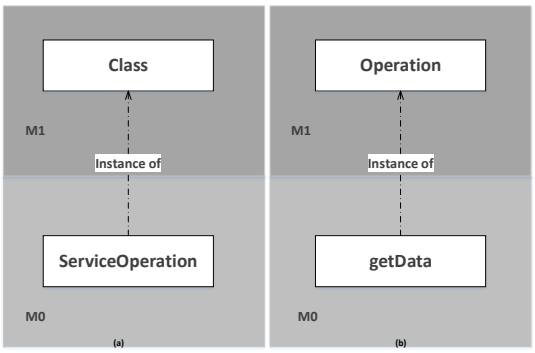

Figure 4.2: The UML two levels representation of the *Class*, *serviceOperation*, *Operation* and *getData*.

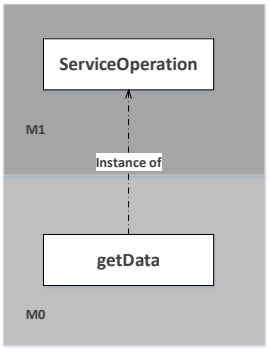

Figure 4.3: The UML two levels representation of the *ServiceOperation* and *getData*.

A similar problem exists in case of resource specification because more than two levels of specifications are needed to completely elaborate the concept of resource. If we consider the CPU resource then the concepts of *AbstractResource*, *CPUResource* are the two levels of abstraction normally modelled but having different type of CPUs for different computer architectures expects that *CPUResource* can have further instances to represent CPU for each platform. Therefore, the concept of *CPUResource* can not be completely modelled using a two level modelling approach and requires it to represent both an instance and a class itself so that it can link the instances at lower level to higher abstraction at upper level.

Although moving *ServiceOperation* one level up solves the initial issue of multiple level modelling, that is not the only challenge that needs to be addressed. The operation *getData* is a conventional operation, and the validation of its behaviour as being the same as a *ServiceOperation* is not straightforward. The behaviour of an operation is defined by its various states and transitions between those states, therefore state machine model of an operation is a key to mapping its behaviour so that it works like *ServiceOperation* as specified in QML/CS *response_time* context model. More discussion about this challenge will take place in Chapter 5.

4.3 Deep Meta-Modelling

Contrary to the constraints of OMG's four-Layer hierarchy like UML, the deep meta-modelling provides an alternative approach to allow multiple levels of modelling so that a system can be modelled using any arbitrary number of levels. That is why the deep meta-modelling is inclined towards giving user the liberty and control on how many different levels are needed for specifying the system under consideration [31].

The difference between the two is not just that deep meta-modelling allows an arbitrary number of modelling levels; the difference lies in the way abstract concepts are mapped to their instances because the instances can be mapped to types too. Another difference lies in the ontological and linguistic concepts and their usage in two-level and deep meta-modelling. The concepts of ontology and linguistic are mixed in two-level because there is no clear identification of the difference between them and the relationship of inheritance is used to establish relationship between the two levels [8]. The deep meta-modelling uses clear ontological and linguistic divisions so that the relationship within the logical and physical domain can be clearly defined. The linguistic structures support the ontological concepts by defining their relationship across multiple levels while keeping their ontological relationship at the domain level. The Fig 4.4 shows a difference of concept for both linguistic and ontological structure when we define a relationship between two concepts in the system taking a cars system as an example.

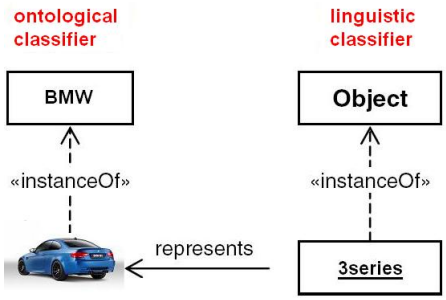

Figure 4.4: Ontological and Linguistic Classification

There are a number of multi-level meta-modelling approaches like [7,29,46,75,104,105] that focus on using multi-level strategy to solve the modelling problems of systems. The approaches indicated that the specification language should be capable of providing information about knowledge elements of that system and the relationship between those elements. Also it can help in reducing the specification ambiguity and the modelling concepts are clearly visible based on the context of their use. Telos [75] is one such example that is a language designed around specification of information and knowledge based system. It has ability to specify representation of knowledge from different domains and comprehensively covers application domain, user models, software requirement and design and methodologies. A prototype object manager ConceptBase [51] was developed based on the Telos object model to evaluate requirement of infinite meta-class hierarchy to extend evolve the specification schema. The ConceptBase extended the meta-modelling to be able to use instantiation of models without limit of modelling depth but it missed the elements organisation and levels

for meta-model.

Dahchour [29] discussed that an effective use of materialization concepts can help in meta-modelling when combined with meta-class approach of TELOS data model. This is possible because the concept of materialization establishes relationship between the entities considering their categories as abstraction and concrete objects as instance of those categories. Additional constraints can be added to the meta-classes, classes and meta-attributes so that the modelling of entities can be facilitated with constraints defined on them. Current TELOS data model does not support providing comprehensive constraint expressions therefore they extended TELOS data model to allow definition of additional constraints on the entities and their attributes. This concept relates to a typical type-object pattern discussed in [31] that also mentions about relationship between two entities. In [104], Varro and Pataricza propose a visual but mathematically precise framework (VPM) that uses the concepts of mathematical definitions and graph transformations to define the abstract syntax and dynamic semantics respectively. This framework uses refinement calculus as a rule to build models and their hierarchies so that multi-level meta-models can created for a system. This framework applies the concept of set theory on the MOF UML constructs so that dynamic meta-model can be used incorporating those concepts. This dynamic nature of meta-models can then be extended to be able to transform the relationship between entities to relationship between meta-models so that these relationships are realised across levels and they provide context-specific meaning to an entity.

Deep instantiation supports the achievement of deep characterisation that facilitates defining entities and their properties to have relationship across levels compared to shallow characterisation available in object-oriented classification semantics. In [60], deep instantiation is discussed with the use of clabject and potentcy as a way to implement deep instantiation in multi-level modelling. They chose Java as the language and did a compiler that would check the deep characterisation incorporated through deep instantiation in Java language. The concept of Clabject [46] originated from the notion that each model element has both an instance facet and a type facet, which are equally applicable. Both facets are categorised as follows:

- A class facet has a name, attributes and relationships; and
- An object facet has values and links.

In order to move beyond the usual 'two-level' paradigm, it is necessary to provide modelling concepts that can be uniformly applied across all levels, in a multi-level classification hierarchy. The concept of potency is to control the impact of a property or attribute to next levels and therefore limiting the models for what should be mapped and what can be left without worrying about relationship between the

models. It can have value of 0 or more, with 0 mentioning completely one level entity definition with no link to level down where as more than 0 value indicates how far down the multi levels this property needs to be modelled in this entities. The Fig 4.5 elaborates the concept of potency by making the entities and their properties available in next level based on their potency value. For example the property *taxRate* has potency 1 and *price* has potency 2; this means that the property *taxRate* will be available on one more level down where as the property *price* will be available two more levels down in hierarchy.

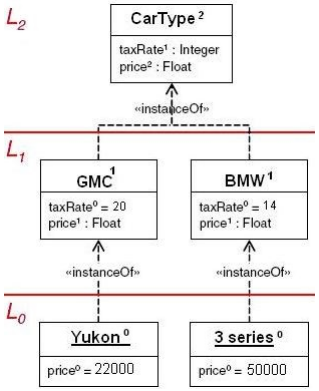

Figure 4.5: Deep Instantiation

Powertypes [46] is another way of achieving multi-level solution that uses the concept of sub-typing to create relationship between entities at different levels. The use of metaclasses and their respective associations to make relationship at different levels creates too many meta-classes and therefore reducing clarity of the design. Also the powertypes are limited to next two levels only because each powertype can characterise features only through the subtype, which itself can not be a power type. There are some languages and frameworks like Nivel [7] and "Open Meta-Modeling Environment" (OMME) [105] that implement deep instantiation. However they do not support multiple ontological types and meta-modelling facilities across levels and therefore limiting the deep meta-modelling capabilities.

We have chosen *Clabject* as a technique to support deep meta-modelling because it is more close to the object-oriented classification semantics and it redefines the class and object concepts so that they are linked across modelling levels. The *Clabject* also overcomes the concerns in *Powertypes* by not having too many meta-classes and therefore the number of associations are also controlled keeping the design very clear. The concept of Clabject is also not limited to two or more levels and can be extended to be used to any number of levels because each entity has a dual facet and therefore

can be used at every level. The modelling problem of Library item mentioned earlier in Section 4.1 can be solved by applying *Clabject* technique so that each category and subcategory in the library item hierarchy is modelled having a double role based on the context. The proposed solution presented in Fig 4.6 indicates how the concept of Java 8 can act as a class and instance at the same time and further levels of modelling can also be added using the same approach where needed.

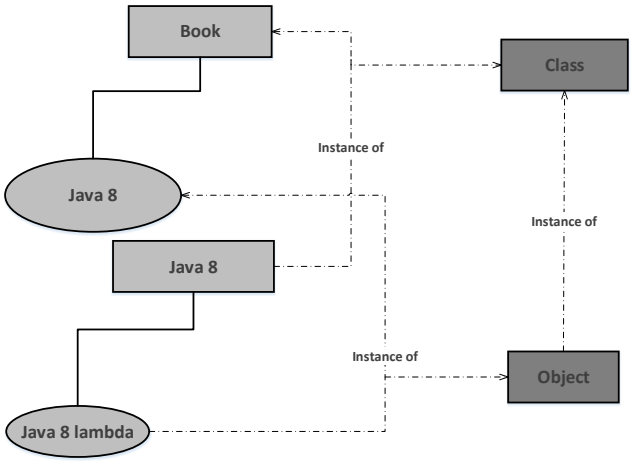

Figure 4.6: The *Clabject* representation of the of Library Item.

Applying the same technique to solve the modelling challenge of our language QML/CS where *serviceOperation* acts as a type and an instance at the same time based on where it is being used in relation to the class or the *getData* method. The *Clabject* representation of a proposed solution to solve the multiple level modelling problem is shown in Fig 4.7. We can see from this figure that the right part shows the modelling problem of UML with two levels of representation and left part shows the same problem addressed using *Clabject* technique where *serviceOperation* is represented as both class of *getData* and as an instance of the class.

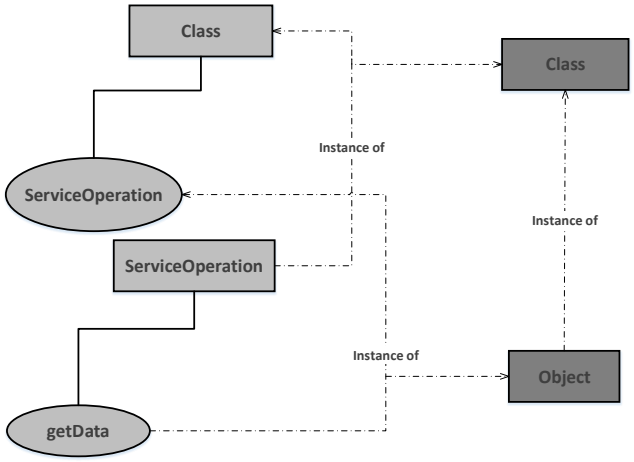

Figure 4.7: The *Clabject* representation of the *serviceOperation* and *getData* derived from [46].

4.4 QML/CS Definition with MetaDepth

Deep meta-modelling is used to simplify the language definition and to automate the maintenance of consistency between classes and objects. Fig 4.8(b) shows the multilevel solution. Compared to 4.8(a), this solution allows the type (serviceOperation) to be a type and an object using the *Clabject* concept [46]. Meta-depth is a framework for deep meta-modelling developed by de Lara and Guerra [30]. It permits the representation of a meta-model to allow an arbitrary number of ontological and linguistic levels and the dual instantiation. MetaDepth allows the construction of a domain specific language as textual modelling. At different levels of the ontologies, the derived attributes and constraints can be specified and evaluated. In addition, MetaDetph supports the Epsilon Object Language (EOL), which allows the user of this too to place some constraints. The constraint and evaluation of the derived attribute levels are determined by the potency concept.

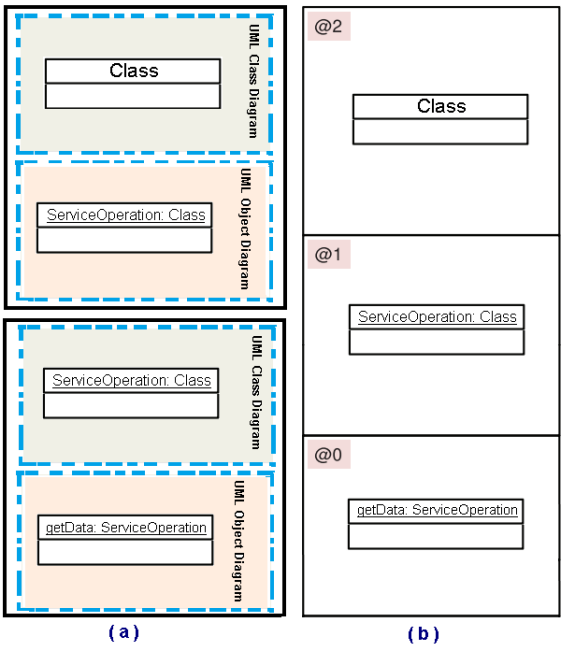

Figure 4.8: Modelling *ServiceOperation* and *getData* using Deep Meta-modelling.

Listings 4.2 to 4.4 show how the three models in Fig 4.8 and Fig 4.9 are defined with the textual syntax. The idea is specifying a three-level meta-modelling architecture where the top-most level contains the definition of meta-classes. Listing 4.2 shows meta-model for QML/CS containing meta-classes of some of the concepts like Measurement, Parameter, Type and Class. It shows creation of a model using the abstract Node *Type* that has same potency level as the model itself, which is 2. The top level *Meta-model* is assigned potency 2, which means that we could create its instances two meta-levels further down. The node *Type* is an abstract node that can be extended by *Class* and *DataType* nodes. *Class* is a node that can have a number of *attributes*, *operations*, *associationEnds* links to *Association* node and class that loops back to the class itself. The node *Parameter* has a name and a type that references *Class* node. The node *Measurement* has a name, formal parameters that references to *Parameter* node and dateType that references *DataType* node. The idea is that the model is created with abstract and concrete nodes with each node having a potency level so that instantiation of the types and nodes can be controlled and also that the model allows specifying the *Clabjects* to allow deep modelling. The node Class allows arbitrary number of attributes so that it can represent any measurement that may hold zero or more helper variables. The node *Class* also supports zero or more operations so that it can represent any number of operations associated with a component providing a specific service. The reference *classes* links back to *Class* itself to establish the connection between two instances of the same *Class*. We present only

the most salient parts of meta-model definition for QML/CS with *MetaDepth* here, the full specification can be found in Appendix B.2.

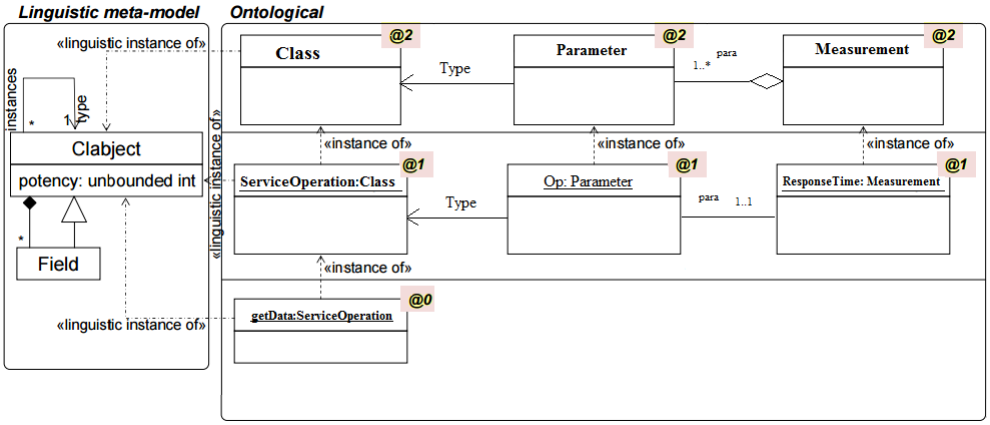

Figure 4.9: Deep Meta-modelling derived from [30]

```
1 Model QMLCSMetamodel@2 {
2  abstract Node Type {
3   typeName:String{id};
4   isAbstract@1:boolean=false;
5  }
6  Node Class: Type {
7   name:String{id};
8   classes:Class[0..*];
9   attribute:Attribute[0..*];
10  operations:Operation[0..*];
11 }
12 Node DataType: Type {
13  name:String{id};
14 }
15 Node Parameter {
16  name:String{id};
17  type:Class;
18 }
19 Node Service{
20  name:String{id};
21  servoperations:Class[0..*];
22 }
```

```
23  Node Measurement {
24   name:String{id};
25   formalParameters:Parameter[*];
26   dateType: DataType;
27  }
28 }
```

Listing 4.2: A meta-model for QML/CS in Meta-Depth

In this way, in the next meta-level we can build a model named *System* of QML/CS meta-model and it is assigned to potency 1. *Complete System* model declares five models (i.e *Service*, *ServiceOperation*,*op*, *Real* and *ResponseTime*) of meta-classes in the meta-model as shown in listing 4.3. It shows the instantiation of *serviceOperation* and *service* from *Class* node, *op* from *Parameter* node and *ResponseTime* from *Measurement* node. The *ResponseTime* Measurement has a parameter named *op*, dataType of real and its parameter references to type *ServiceOperation*. This level of meta-model can instantiate only those nodes from upper level that are defined and have a potency level to be at least 1 so that they can be referred here. It is important to mention that potency level at upper level does not control how further down the instances at this level can be used so we need to mention the potency level here to control the instantiation of these concrete representations.

```
1  QMLCSMetamodel System@1 {
2  Service SystemService {
3   name="SystemService";
4   serOp: ServiceOperation [0..*];
5  }
6  Class ServiceOperation {
7   name="ServiceOperation";
8  }
9  Parameter op {
10   name= "ResponseTimeop";
11   type: ServiceOperation[1]{type};
12  }
13  Measurement ResponseTime {
14   name="response_time";
15   opParam: op[1]
16   dataType: Real;
17  }
18 DataType Real{
19   name="real";
```

```
20 }
21 }
```

Listing 4.3: Model Definition for QML/CS in MetaDepth

In the bottom meta-level (as shown in listing 4.4), a specific example is specified based on the *System* model so that the relationship between the model elements and a specific instance of the model is clear. It also indicates the mapping between node representations and the concepts of a real system. To understand that the problem of dual representation of the modelling concepts is solved, this level shows the relationship between *ServiceOperation* and *getData* as they both represent an instance-Of entity and we represented *ServiceOperation* in the listing 4.3 as instance of *Class* which was not possible to create using standard modelling language like UML.

```
1 System CounterApplication{
2  SystemService counter {
3   name="counter";
4   serOp=getData;
5  }
6  ServiceOperation getData {
7   name = "getData";
8  }
9  op getDataopParameter {
10   type = getData;
11  }
12 }
```

Listing 4.4: Instances *responseTime* context Model Definition for QML/CS in MetaDepth

4.5 Summary

This chapter discussed the problem of dual instantiation and provided a solution so that the entities at different levels can represent the same entity based on the context of their use. It provides a base for deep modelling and the entities can be used at as many levels as intended with control of what parts of it to be available at each level. This gives us liberty to use complete or partial entities as per modelling requirement and linking between the entities can be established based on their expected relationship. As shown in the listings presented in this chapter, metaDepth tool can be used to implement deep modelling for QML/CS specification. The concepts of Measurement, Parameter and *ServiceOperation* are represented as Node in metaDepath and

relationship between them is modelled using attributes and associations.

The next chapter will present the second problem of mapping a measurement to a concrete application, and shows its solution of specifying mappings between context and application models.

5

Specifying Mappings between Context and Application models

The separation of context and application model specification from their concrete implementation in a specific application domain allows maintenance and reuse in component-based system and modelling NFPs [110]. This gives liberty to specify measurements independent of the concrete application. At the same time, having different models (each one describing a certain behaviour of the system) requires their validations when applying measurement to represent an appropriate behaviour in target application. The technique of weaving model is the proposed solution to address this issue of models validation and compatibility.

This chapter describes how can weaving model be used for validation of mapping between context and application models conforming to meta-models mapping that is derived from QML/CS meta-model. Section 5.2 discusses the problem of applying measurement to concrete application. Section 5.3 shows weaving models to specify mappings between the context and application models described during the specification of NFPs of component-based applications. Section 6.4.2 explains rules that have been used and proposed to provide a guideline in validating the mapping.

5.1 Introduction

Applying a measurement to concrete application plays an important role for exhibiting the non-functional aspect of a software system and provides measures of NFPs in a specific working environment. Since the measurement is applied to a specific operation of the target application, the behaviour of the operation should be same as theof expected behaviour of relevant measurement parameter so that a standard operation can be transformed to measurement parameter. The abstract specification of measurement that is based on its context model is independent of the target operation to avoid any coupling or dependencies on the target application so that the

measurement can be applied to any generic system where application model follows the specified context model for the measurement. To document and validate this, there is a need to make the mapping explicit through a separate model.

Mapping between application and context models can be referenced in measurement's argument providing the name of the mapping model. The mapping is defined by a mapping model that clearly describes structure of the mapping to link different features of both application and context model. The application of a measurement to an application needs detailed description and definition of the links and the rules to validate that the mapping link is valid. This process is referred to as weaving model [6]. It is employed based on some pre-defined rules to ensure formal and complete mapping between those models. While specifying the mapping, the definition provided in [49] is considered, which is "*the mapping description may be in natural language, an algorithm in an action language, or a model in a mapping language*". The mapping between context and application models are defined using a mapping model and built by weaving those models in a set of rules that also implemented as mapping functions in QML/CS.

5.2 Applying a Measurement to a Real Application

Applying a measurement to a concrete application is not straightforward to be achieved. It is important to check that a type of the parameter of a measurement matches the type of operation that is being provided as a concrete argument to a measurement in the concrete application. The specification of measurement in the meta-model and the implementation of operation for which measurement is being applied is different in the sense that the parameters of measurement may behave differently from traditional method that is part of a component providing a specific service. The problem is how we capture the behaviour for both the representations of measurement in context model and the operation in application model so that a link can be established between them based on their compatibility with each other. Fig 5.1 shows a simple example where a method named *getData* that is instance of *Operation* in UML model needs to be mapped as a concrete argument to *ResponseTime* measurement. Passing this method requires that it must satisfy the measurement definition that expects *responseTime* parameter to be of type *ServiceOperation*. *Operation* is associated with a state machine that defines its behaviour and the type of measurement's parameter *ServiceOperation* is also associated with a state machine that represents its behaviour.

Therefore, a mapping between *Operation* and *ServiceOperation* classes and their state machine models should be established so that compatibility between their behaviour

can be evaluated. To be able to map state machine models of both *Operation* and *ServiceOperation*, the states and transitions of both state machine models must map based on some predefined condition. The aim of specifying the mapping between context and application models is to show that any given operation behaves the same as the behaviour of measurement's parameter.

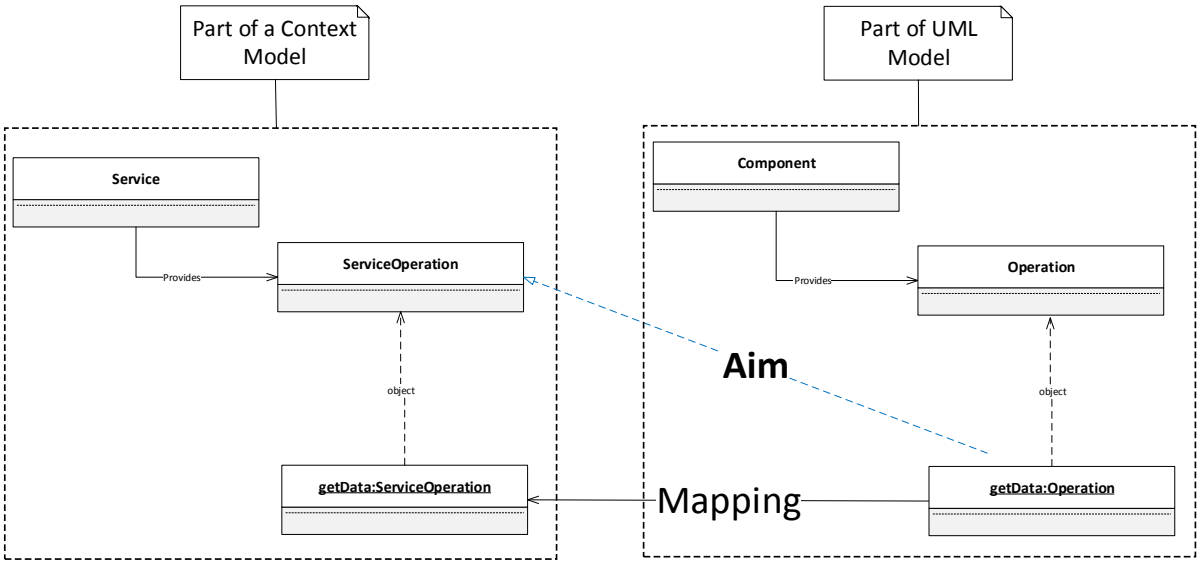

Figure 5.1: An Example of the Mapping between Application and Context Models

5.3 Weaving Model

The concept of weaving models [6] is to define a relationship between a source model and a target model with certain mapping conditions based on predefined rules which can be user-defined. Weaving model contains a set of links between elements of both the models between which it is establishing mapping [33]. Fig 5.1 shows the idea behind mapping and what the ultimate objective is that should be achieved as a result of mapping. The right side of the figure shows a part of the UML model that represents application model of an operation *getData* that is part of a component and measurement is being applied to this operation. The left side of the figure shows part of context model for the measurement being applied to *getData* and indicates the type *ServiceOperation* as a representation of the parameter for this measurement. The link arrow that goes from *getData* of the UML model to *ServiceOperation* in context model identifies the objective of mapping *getData* to behave like *ServiceOperation* so that the measurement context model can be applied to it. Since *getData*

in the UML model is an instance of *Operation* class therefore the ultimate mapping should exist between *Operation* and *ServiceOperation* classes.

We introduce mapping strategies that help in mapping the application to context model. Fig 5.2 shows a high level representation of a mapping model that indicates how different concepts of mapping are represented and the relationship between them is also identified. The different meta-classes like *ClassMapping*, *StateMachineModelMapping*, *StateMapping* and *TransitionMapping* represent link between application model and context model. It mentions how these meta-classes should be mapped for both the models and how they are all linked and controlled by a common concept of meta-model. The meta-class *ClassMapping* controls the mapping between source and target classes in application model and context model respectively. It is important to note that each *ClassMapping* should have a link with *StateMachineModelMapping* to determine that class belongs to specific a sate machine model. The meta-class *StateMachineModelMapping* controls the mapping between source and target state machine model that contains a list of all the states and transitions that belong to the classes in the application model and context model. The meta-class *StateMapping* controls mapping of each state in state machine model of the application model to a state in state machine model of the context model as required by the *StateMachineModelMapping*. The meta-class *TransitionMapping* controls mapping of each transition in state machine model of the application model to a transition in state machine model of the context model as required by the *StateMachineModelMapping*.

Fig 5.3 provides an example of mapping between *Operation* class and *ServiceOperation* class based on the state machine model associated with both classes. The figure shows what different states exist for the context model of a measurement and these states, along with transitions in each state, are mapped to states and transitions in the application model to ensure compatibility between them. This typical example mentions states Idle, *RequestAvailable* and *HandleRequest* that represent a measurement being called and processed along with the initial *Idle* state to show when it is waiting for request or done with the request processing. The transitions like *RequestArrival*, *StartRequest* and *FinishRequest* are associated with these states and their impact on changing state from one to another is also shown in this figure. These sequence of transitions plays an important role in the mapping because each step or transition has variables attached with it and provides a guideline on what value should be given to the variable. Different mapping strategies are shown in this figure that establish the mapping model between different elements of application and context model. For example, mapping strategy *ServOpMapping* references to source *Operation* and target *ServiceOperation* classes. Another mapping strategy *StateMachineModelMapping* provides link between state machine models of the classes being mapped. For each state and transition of these state machine models, a *StateMapping* and *Transition-*

Mapping strategies are defined to establish mapping between states and transitions of source and target models. The weaving model uses those mapping strategies based on pre-defined conditions that establish the simulation relationship and validate that *getData* has a state machine model that is mapped to a state machine model associated to *ServiceOperation*. In addition, it also needs that the states and transitions of both state machine models confirm to an accepted behaviour of an operation *getData* to behave like *ServiceOperation*.

5.4 Validation

The mapping between classes, state machine models (including states and transitions) in context and application model is valid only if it ensures that the following rules are satisfied. These rules have been proposed to provide a guideline in mapping of state machine models to validate that behaviour of source and target operations is similar [110]. The symbols and legends used in the equations are as follows: $\forall$ - It denotes for all expression, which means that every element of the set should validate to the condition being used in the equation or expression. $\exists$ - It denotes to the existence expression, which means that there should be at least one element in the set that should validate to the condition being used in equation or expression. $\sum$ - This indicates to the set of states of a state machine. $\sum_{C}$ - C indicates context model and this $\sum_{C}$ means all states in the context model. $\sum_{A}$ - A indicates application model and this $\sum_{A}$ means all states in the application model. T denotes transitions in context or application model.

1: Each state in application model is mapped to at least one state in the context model. The *map* function in the equation indicates an expression to check mapping between the two states based on the mapping model.

$$\forall s1 \in \sum_{A} \cdot \exists s2 \in \sum_{C} \cdot map(s1, s2).$$

This condition is expressed in OCL as shown in listing 5.1. It uses built-in functions of OCL like *forAll* and *exists* to represent the expression mentioned as mathematical equation. This OCL script shows a better readable expression. The line 1 shows the link to mapping meta-model where as line 3 defines an invariant for evaluation of the state mapping to be either true or false considering states in both context and application models.

2: Each transition in application model is mapped to at least one transition in the context model. The *map* function in the equation indicates an expression to

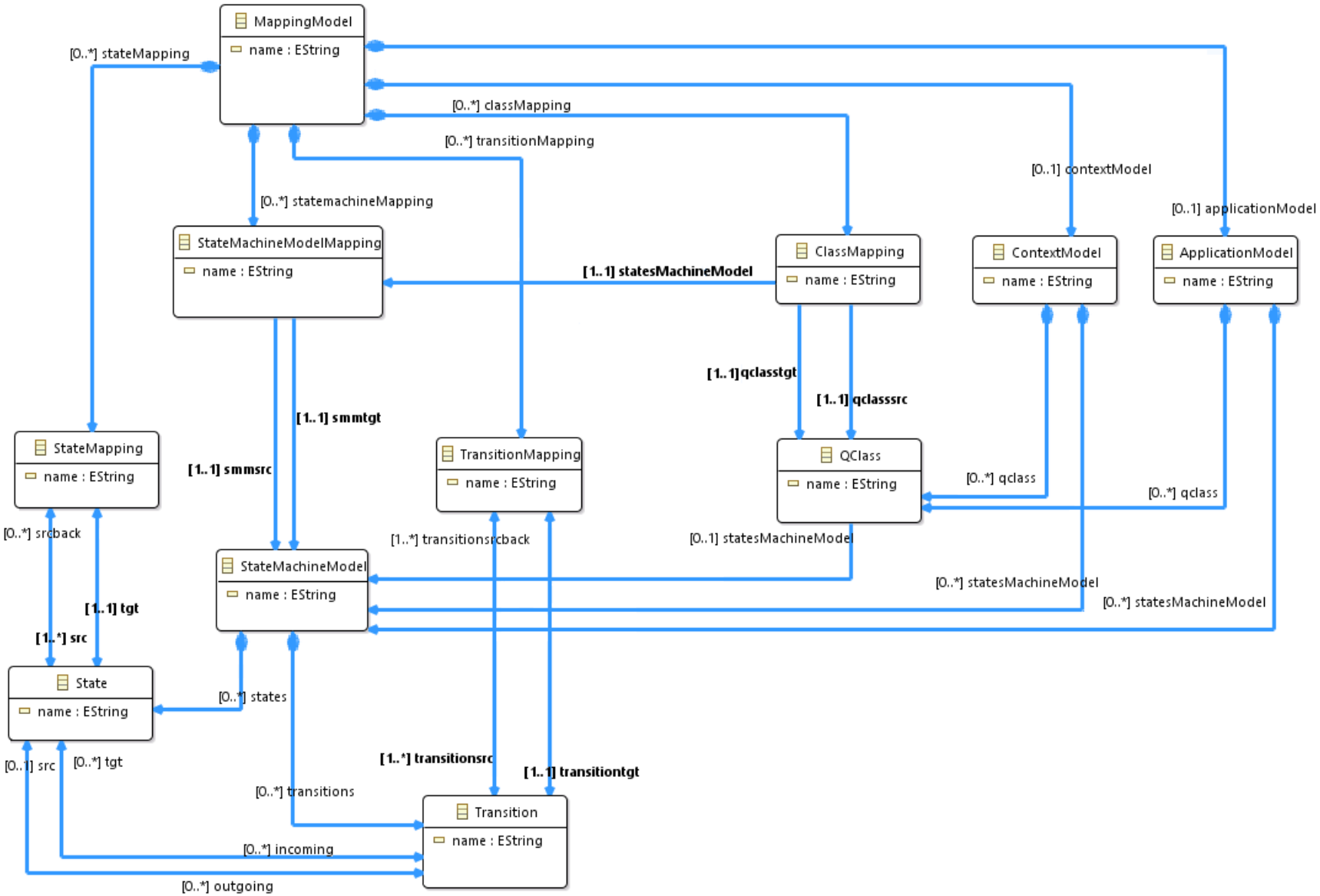

Figure 5.2: Mapping Meta-Model.

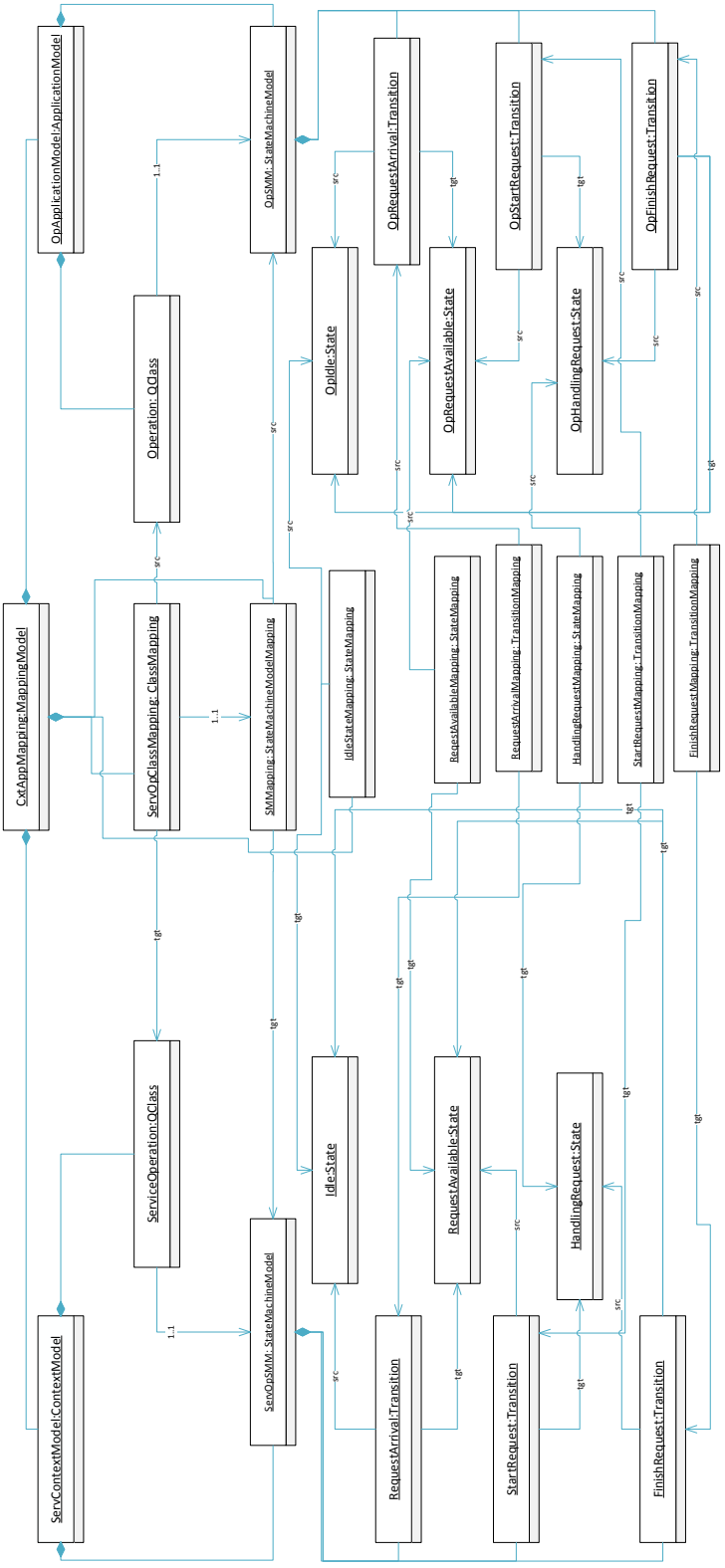

Figure 5.3: Example: Mapping Model between context and application models.

```
1 context MappingModel
2
3 inv: stateMachineModelMapping->forAll(
4    smmsrc.states->forAll(s1 |
5     smmtgt.states->exists(s2 |
6       stateMapping->exists(mapping |
7         mapping.src->includes(s1) and mapping.tgt->includes(s2)))))
```

Listing 5.1: *Each state in the application model is mapped to at least one state in the context model*

check mapping between the two transitions based on the mapping model.

$\forall t1 \in T_A \cdot \exists t2 \in T_C \cdot map(t1, t2)$.

This condition is expressed in OCL as shown in listing 5.2. It uses built-in functions of OCL like *forAll* and *exists* to represent the expression mentioned as mathematical equation. The line 1 shows the link to mapping meta-model where as line 3 defines an invariant for evaluation of the transition mapping to be either true or false considering transitions in the state machine model of both context and application models.

```
1 context MappingModel
2
3 inv: stateMachineModelMapping->forAll(
4    smmsrc.transitions->forAll(t1 |
5     smmtgt.transitions->exists(t2 |
6       transitionMapping->exists(mapping |
7         mapping.transitionsrc->includes(t1) and mapping.
    transitiontgt->includes(t2)))))
```

Listing 5.2: *Each transition in the application model is mapped to at least one transition in the context model*

3: The third condition is important to ensure that the mapping is consistent, and this is achieved via establishing a simulation relationship.

$\forall s1 \in \sum_{A} \cdot \forall s2 \in \sum_{A} \cdot \forall t1 \in T_A \cdot trans(t1, s1, s2) \Rightarrow \forall s3 \in \sum_{C} \cdot map(s1, s3) \Rightarrow \exists s4 \in \sum_{C} \cdot \exists t2 \in T_C \cdot map(s2, s4) \wedge map(t1, t2) \wedge trans(t2, s3, s4)$.

For any two states s1 and s2 in the application model linked with a transition t1, there exists at least one state s3 that can be mapped by s1 and at least

one state s4 that can be mapped by s2. Also the transition t2 that connects s3 and s4 in the context model should be mapped by t1. Only then it satisfies the mapping between the application model and the context model. This condition is expressed in OCL as shown in listing 5.3. The listing shows how first two conditions have been joined to evaluate mapping of a each pair of states in the application model to at least one pair in the context model. The equation comprehensively considers different elements of the mapping so that mapping within the states of application model and context model is checked before the states and transitions in those models are checked for mapping between them.

```
1  context MappingModel
2
3  inv: stateMachineModelMapping->forAll(
4      smmsrc.states->forAll(s1|
5        s1.outgoing->forAll(t1|
6          t1.tgt->forAll(s2|
7            s1.srcback.tgt->forAll(s3|
8              s2.srcback.tgt->exists(s4|
9                s3.outgoing.tgt->includes(s4)
10               and
11               s3.outgoing->select(t2|
12                 t2.tgt->includes(s4))->forAll(t2|
13                   t1.transitionsrcback.transitiontgt->includes(t2))
14                   ))))))
```

Listing 5.3: *The mapping consistency condition*

5.5 Summary

This chapter presented mapping between the application and context models with the help of a mapping meta-model that defines link between each meta-class of the context model to concrete classes in the application model. We elaborated how each model is represented by a class and associated state machine model and then established mapping between those states and transitions that constitute the state machine model. The rules were set out to control the mapping so that illegal mapping could be avoided and the states and transitions from source are mapped to only compatible states and transitions in target.

As mentioned above, the problem of specifying mapping between context and application models become crucial when applying measurement to concrete application. In this chapter, we have shown how we addressed this problem by making the use of weaving models [6], have defined the meta-models according to the different MDA abstraction levels and demonstrated how mapping between context and application

models can be achieved. The mapping strategy, with the help of weaving model, ensures that the mapping does not allow any illegal mappings that could break the rules for mapping as it is controlled to micro level of states and transitions. The QML/CS language hides all this abstraction from the user and therefore provides a fairly good chance that wrong mappings are avoided for which there is no formal way to avoid wrong mappings in general. We provided a generic mapping model, which not only loads as input the application and context models, but also establishes the weaving model based on pre-defined conditions. Thus, based on the weaving model employed, we can check that a behaviour of the parameter of a measurement validates the behaviour of operation that is passed as a concrete argument to a measurement in the application. As a result, it is possible to check and validate operations in concrete applications.

Next chapter will provide implementation of QML/CS and highlight problems faced during implementation and how they were addressed.

6 Implementation

This chapter presents implementation details of QML/CS language[1] and provides technical details of how different elements of QML/CS work together to provide a specification language for NFPs. It also provides implementation insights into how the solutions presented in the chapters 4 and 5 are integrated as a complete definition of QML/CS.

The purpose of implementation is not just to provide a generic QML/CS language but also an environment that can be used to write specification for NFPs of component-based systems. The implementation contains a validation component so that QML/CS is validated to comply with language syntax, semantics and adherence to the rules specified in context model. *MetaDepth* [30] is a framework for deep meta-modelling that can be used to write specification as discussed in chapter 4 but since it does not have full support for comprehensive parser, it can not be used as a language editor. Therefore, we decided to use Xtext [37] that comes with customizable user-interface and parser components. Xtext can be used to create an Eclipse-like IDE for the domain specific language(DSL) being developed. It provides good support for grammar, validation of grammar against customised rules, validating imported models, and integrated IntelliSense to facilitate rapid development. However, the existing parser generators like Xtext are restricted to have only two level (M2 and M1) models within a meta hierarchy and therefore do not provide the flexibility to define multi-level/deep meta-modelling [31,105]. As extending parser generations with deep meta-modelling is well beyond the scope of this thesis, we have chosen an alternative lifting approach [111] to be able to apply deep meta-modelling. The implementation of this approach will be discussed in Section 6.3.

[1]The implementation link of our tool: http://a-alreshidi.github.io/QML-CS/. Screenshots of our language can be found in Appendix A.4

6.1 Technologies

As part of this thesis, we implemented a EBNF based grammar and Xtext based compiler that will validate the grammar and allow the user to be able to specify the language and Xtext can validate it based on the rules specified. This helps in generating run-time implementation of QML/CS constructs. The Xtext based QML/CS compiler builds a parse tree for the language and then creates an Ecore file that has complete hierarchy of the QML/CS language concepts derived from the grammar. It also creates a gen file that is used to create an editor that can be used to write grammar syntax based language specification. The focus of the implementation was to develop the language and provide an integrated environment so that the language user can use that environment to write the specification according to language rules. Following tools, technologies and standards were used in development.

6.1.1 Eclipse

Eclipse [36] is an open source community that manages many projects. Lot of individuals and organizations contribute to the projects developed under this umbrella and they range from small to large projects including commercial level projects. The Eclipse platform is one of major contributions of this Eclipse community that provides IDE for software development and it has many flavors for different languages and technologies. The architecture of Eclipse platform is very flexible and built on plugins so that different plugins can join together to produce a customized version of Eclipse platform for a specific language based software development. It aligns perfectly with a component-based paradigm where the base kernel is responsible for loading all the required plugins and they are instantiated when needed. The plugins work as an extension to base platform in many different ways including providing some extra functionality themselves or work as a link to other plugins to be used. The Eclipse platform allows the users to develop their own plugins as well on top of a large collection of plugins already developed contributed by different people around the globe.

Eclipse is used as base platform for our language specification and implementation where we implement all components of QML/CS language. The components mentioned in next sections like EMF and Xtext are used from within Eclipse and the final editor for the language is also opened in Eclipse interface.

6.1.2 EMF

The EMF, stands for Eclipse Modelling Framework [99], is modelling framework project from Eclipse community that allows development of tools based on structured

data models and provides model editing as well as code generation facility extending the basic Eclipse platform with required plugins. It understands the models specified in XMI and then provides tools to convert the model into implementation classes as well as editing the model that updates both XMI and the generated classes. It supports three levels of code generation called Model to produce java interfaces, Adapters to generate implementation classes and Editor to produce structured editor that will provides a basic editing facility for models.

EMF is used to support the generation of language editor as Ecore and gen files. The Java classes are derived from gen file. It also helps to create instances for context model and application model from meta-model so that they can be imported in the qmlcs editor while writing the specification.

6.1.3 Xtext and Xtend

Xtext [14] and EMFText [44] are two language development frameworks that support implementation of domain specific languages. Both create the output as Eclipse plug-in that can be used to implement the language. EMFText allows users to define text syntax for languages modelled by an Ecore meta-model. It requires the users to be well versed with EMF, as they need to build a model of their domain specific languages. Xtext was originally developed as part of openArchitectureWare [oAW] project initiative and later added to Eclipse community. It covers all aspect of a programming language so that user can customize and implement the language infrastructure including parsers, linkers, compilers and interpreters so that the IDE provides integration of all these components. Xtext has advantage over EMFText that it does not need EMF model to be created by user and generated it from the grammar automatically. The user specifies the grammar for the language and Xtext created EMF model from it automatically and generates implementation classes. It is very flexible because it conforms to the Google Guice [37], a lightweight dependency injection framework that allows using of existing DSL and API implementations as well as smooth replacement of default implementations with user developed DSL implementation. It provides a complete running language based on the abstract syntax we define for domain specific language. It is also capable of understanding both textual and tree based models specification and then create parser and language editor for the models integrating the compiler or interpreter to execute the script written in developed language. The scope of the language features and linking them with validation are key parts of an environment being used to write the specification. Xtext has a scope provider component that helps in writing the specification upon user request and completes the contents based on the scope of the content being used and its linking with the grammar. We have used Xtext scope provider in our implementation to facilitate the user to write specification and automatic content completion.

Xtend [14] is a standalone project managed by Eclipse community that was initially developed as part of Xtext framework. It is an advanced and general purpose programming language that is based on Java language and uses many features from the language along with extending it with operator overloading, extension methods and type reference. Although it is an object oriented language but it has support for lambda expressions of functional programming extending the language with advanced and complex expressions to be mentioned. The code written in Xtend is compiled to Java files and therefore can be linked with any Java libraries and also integration of the output as a jar package is possible with any Java application. Although scope provider is part of Xtext but the customisation is possible and Xtend allows us to write scope providers based on the requirement.

6.2 QML/CS Component Architecture

The relationships between different components in QML/CS prototype can be seen in Fig 6.1. Subsystems can be categorized as follows:

- Component *QML/CS Grammar*: The component provides complete definition of QML/CS language grammar. It also provides interface for other components to use this grammar as input to generate languages based on this grammar.
- Component *QML/CS Generator*: This component combines all inputs into Java implementation classes and makes the editor component ready to be used for specification of NFPs.
- Component *QML/CS Scope Provider*: This is a Xtext component that provides scope description for the grammar we specify for QML/CS. It helps QML/CS Generator to identify scope for each feature being generated from the grammar and provides linking information between different language features.
- Component *QML/CS Grammar Validation*: This component provides validation of QML/CS grammar, for example, this component can check measurement expression, by validation its the number of measurement's parameter and shows the error *"Number of parameters does not match definition! Expecting"*.
- Component *QML/CS Helper*: This component provides support for loading the required resources and make them available for the other components *QML/CS Grammar Validation* and *QML/CS Scope Provider*. It uses Xtext resource importer to load the resources so that they can be used in specification.

- Component *QML/CS Editor*: The component editor provides contextual content assistance for all the rules described in component *QML/CS Grammar* and it is the interface that is used by measurement and application designers.

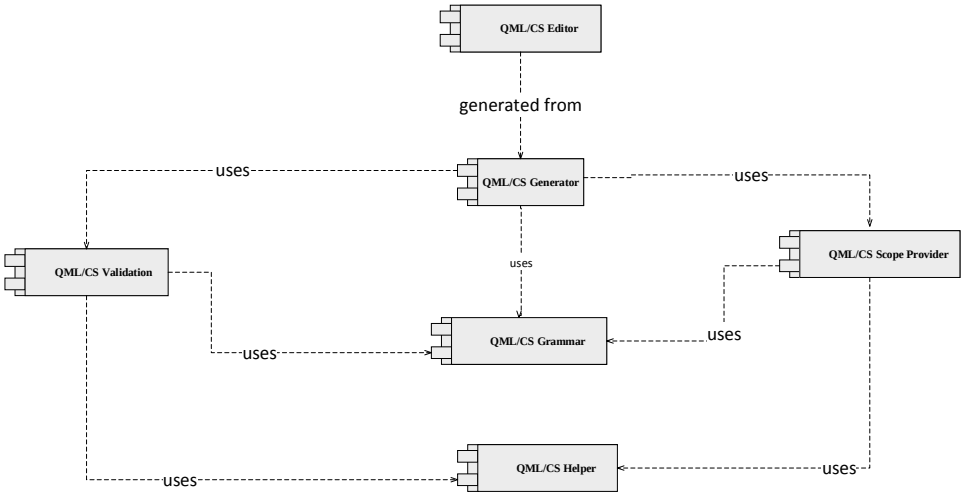

Figure 6.1: QML/CS Component Diagram

6.3 Implementing Deep Meta-modelling

As mentioned in Chapter 4, deep meta-modelling is a solution to the problem of representing a type differently at different levels and *Clabject* assists in achieving this purpose. However, implementing *Clabject* concept is limited by the capabilities of the EMF architectures of QML/CS meta-model like EMF meta-modelling architecture used to implement it because such architectures lack the ability to implement more than two levels (Ecore:M2 and xmi:M1) of meta-modelling. In his blog, Boersma [68] recently discussed the potential issues with multi-level modelling and presented an approach that could address this problem; but only for a two level modelling situation. It is also mentioned that multi-level modelling needs a different meta-level for the type and its instances even though when same definition applies to both the type and its instantiation. It is used *Xtext* DSL grammar implementation as example to define the meta-modelling requirement and then provide a sample possible implementation that would support meta-level modelling concept. It was done by defining named attributes that can link to another entity so that meta-model of both the instance and the entity it belongs to are different. A concept of *EntityInstance* is introduced as an abstract concept that points to an Entity and also contains Feature values so

that each Feature value can be linked to the entity it belongs to and this entity is defined one meta-level above. Although it addresses the problem but making it generic solution is a challenge and as mentioned in the blog as well, dynamic expansion of the syntax can be worst if the size of the grammar is bigger. Also it is pointed out that being able to support user defined contexts for grammar definition is not easy to implement for ever expanding grammar expressions.

The lifting approach [111] is a technique that can be used to elevate a model into a meta-model by expanding the domain specific modelling constructs. It is actually an adaptive approach that will elevate the instance into a type for which further instances can be created and therefore the purpose of multi-level modelling can be achieved. This technique fills the gap in existing language and supplements it with new features and structures, therefore, we followed the same idea to address the requirement of deep-modelling but our implementation is different from the perspective that we did not need to create models and their transformations and only checking the model and then its instance so that QML/CS can support deep meta-modelling to create multi-level models. One example of how the Lifting approach takes a model and raises or promotes it to be a meta-model is shown in Fig 6.2. A concrete example of how this elevation works in a real system is shown in Fig 6.3 where an operation *getData* of application model *counter* is passed as concrete argument to the *responseTime*. Then, while validating it against the type of parameter of measurement *responseTime*, it will be elevated and its type checked and validation of the mapping model is performed.

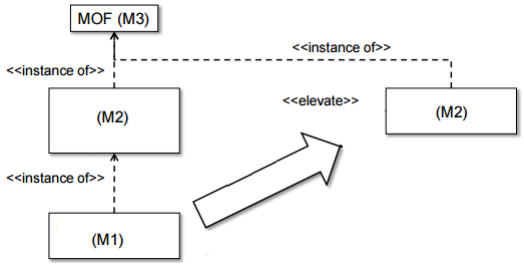

Figure 6.2: Implementing *Clabject* concept in QML/CS derived from [111].

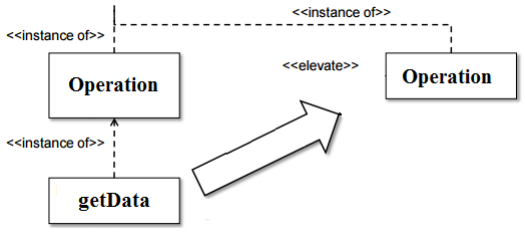

Figure 6.3: *getData* and *Operation* Lifting.

The lifting approach is implemented as part of the function *CheckMeasurementExpression* where every operation in the application model, we define a lifter that takes any operation at model level rather than instance level of the application once it is passed as a concrete argument to the *MeasurementCallExpression* and elevates it by expanding the mapping model constructs and replacing them by its type. Rather than handling the instances of arguments, we have considered the type of the arguments so that the type can be elevated implementing the lifting approach. Listing 6.1 shows the implementation of the lifting approach, which can be seen specifically between the lines 7 and 13. The lines 7 to 9 of this listing indicate how we access the types from measurement expression and measurement declaration rather than instances and this is shown in Fig 6.3 where we access the type *Operation* rather than instance *getData*. A valid mapping between parameters in application and context model is required for the elevation to work and part of that is achieved in methods *mapOperations* and *getMappingModel* references. This listing also checks internal states and transitions of those types to ensure that it can be elevated, which is done in method *mapOperations* as shown in listing 6.2, and it is discussed in more details in Section 6.4.2.

6.4 Implementing Mapping Model

Implementing weaving model is not possible with existing technologies, we had to build Infrastructure to allow the representation of mapping model as these features is not provided by standard technologies. *Xtext* does not have the ability to implement weaving model and validate the inputs to ensure that the input confirms to the mapping model. We have implemented a weaving model feature so that specifying the mapping between application and context model is possible. The implementation contains not only mapping between the models but also rules to validate the mapping between source and target models. It requires implementation of a function that will take both models as input and then validate based on some predefined set of rules by considering internal types represented as state-machine for each model being validated. We show the implementation parts of mapping model. We start by describing the mapping model Infrastructure. Next, we explain how we represent the mapping

```
void checkMeasurementExpression(MeasurementExp measurementExpression) {
  for each argument and for each parameter
    argument := next measurement argument expression
    parameter := next measurement parameter
    if (argument is valid and parameter is valid)
    begin
      retrieve name of argument into _name
      retrieve type of argument into appOperType from _name
      retrieve type of measurement operator into conOperType
      if mapping exists for measurementExpression and argument then
      begin
        retrieve mapping into argumentMapping
        map operations in argument and parameter
        measurement expression is valid
      end
    end
}
```

Listing 6.1: Lifting Approach in Xtext

```
void mapOperations(MeasurementExp exp, QClass appOper, QClass conOper,
    MappingModel mapping) {
  assign state machine model of application model operation into
    appModel
  assign state machine model of context model operation into conModel
  if conModel is not valid
  begin
    throw error and return
  end
  if (appModel is not valid)
  begin
    throw error and return
  end
  check state mapping of application model Operation and context model
    Operation using the provided mapping
  check transition mapping of application model Operation and context
    model Operation using the provided mapping
  check consistency of application model expression and context model
    expression using the provided mapping
  if (no error thrown)
  begin
    mapping exists between operations
  end
}
```

Listing 6.2: MapOperation Validation in Xtext

```
application Counter;
declare service counter{
  provides Operation int getData();
always response_time((getData by Mapping.Mapping1)) < 20;
```

Listing 6.3: Mapping Definition Syntax in QML/CS language

```
MeasurementExp:
  (measurement = [MeasurementDeclaration]) ( '((' '))' | '(('
    ((argument+=MeasurementExpArgument) (',' (argument+=
    MeasurementExpArgument))*)?
  '))')
;

MeasurementExpArgument:
  name = [ApplicationOperation] 'by' mappingRes=ID '.' mapping = ID
;
```

Listing 6.4: Mapping Grammar in Xtext

model and what are the functions used to perform such mapping model. We then discuss every function of QML/CS mapping model validation individually.

6.4.1 Mapping Model Infrastructure

A mapping model infrastructure of QML/CS is presented. It represents different structures of the language and how a mapping model helps in mapping the application model to the relevant context model. This infrastructure helps in building the link between two models. A reference to this mapping model is established via defining grammar rules so that the link is consistent from grammar to implementation. In the model-mapping declaration shown in listing 6.3, the *by* keyword is used to refer to the model-mapping to be used after the name of the operation is defined. It will link the definition with the model-mapping and help to evaluate the mapping and compatibility of the definition against the model. This is achieved through mentioning mapping link in the grammar so that we can specify this, as shown in Subsection 6.4.

6.4.2 Validation Implementation

CheckMeasurementExpression Function: This method is used to evaluate the measurement and ensure that it is mapped to the correct measurement based on its declaration. First of all it checks that the number of arguments passed in the measurement expression and the declaration for that measurement are same. If they are

```
1  void checkMeasurementExpression( MeasurementExp mexpr){
2    retrieve measurement declaration for measurement expression into mdecl
3    retrieve list of arguments of measurement for measurement expression
       into _argument
4    retrieve list of parameters from measurement declaration into _params
5    if no of entries in _argument and _params is not same
6    begin
7      throw error and return
8    end
9    for each entry in _argument and _params
10   begin
11     retrieve current argument into arg
12     retrieve current parameter into prm
13     check mapping between arg and prm
14     if mapping not exists for arg and prm
15     begin
16       throw error and return
17     end
18     check mapping of operations for arg and prm
19     if mapping not exists for arg and prm operations
20     begin
21       throw error and return
22     end
23     measurement expression is valid
24   end
25 }
```

Listing 6.5: Check Measurement Expression Function in QML/CS Validator

not the same, it means the measurement expression is not valid and therefore measurement expression will be rejected and not processed further. Next thing to check is the types of the parameters. If the number of parameters is same, can the types be mapped to each other. If the types can be mapped then it tries to map the operations as presented in listing 6.5.

MapOperations Function: This method attempts to map the operation from application model to the operation in the related context model. It uses mapping model to do this so that only compatible operations can be mapped. It retrieves the application and context model for both operations being mapped and performs state mapping, transition mapping for the states and transition consistency of the transitions to ensure that the operations are mapped only if they are compatible as can be seen in listing 6.6.

```
1  void mapOperations(MeasurementExp exp, QClass appOper, QClass conOper,
       MappingModel mapping) {
2    assign state machine model of application model operation into
       appModel
3    assign state machine model of context model operation into conModel
4    if conModel is not valid
5    begin
6      throw error and return
7    end
8    if (appModel is not valid)
9    begin
10     throw error and return
11   end
12   check state mapping of application model Operation and context model
       Operation using the provided mapping
13   check transition mapping of application model Operation and context
       model Operation using the provided mapping
14   check consistency of application model expression and context model
       expression using the provided mapping
15   if (no error thrown)
16   begin
17     mapping exists between operations
18   end
19 }
```

Listing 6.6: Map Operations Function in QML/CS Validator

```
1  void stateMappingCheck( QClass appOper, QClass conOper, MappingModel
       mapping) {
2    retrieve state machine model of application model operation into
       appModel
3    retrieve state machine model of context model operation into conModel
4    retrieve list of states of appModel into _states
5    get state mapping into _stateMapping
6    for each state in _states
7    begin
8      if name of state matches with state in context model
9      begin
10       retrieve the first mapped state in application model into smap
11       if (smap is not valid)
12       begin
13         throw error and return
14       end
15       retrieve list of states in source parameter into _src
16       for each state in _src
17       retrieve the first mapped state in context model into cst
18       if (cst is not valid)
19       begin
20         throw error and return
21       end
22       mapping is validated
23     end
24   end
25 }
```

Listing 6.7: State Mapping Check Function in QML/CS Validator

stateMappingCheck Function: This method ensures that the states in state machine model for application model and context model can be mapped to each other. It checks that there is mapping for each state in the application model to at least one state in the context model; therefore it satisfies the first condition discussed in Section 5.3. For this it retrieves the application and context model for the operations and then runs a loop on states of application model. It finds the first state mapping in which the set of source states contains the name of the state in the application model. It then checks that the target state of the state mapping exists in the context model. This will check each state in application model and check in context model if there is at least one state that is mapped as can be seen in listing 6.7.

transitionMappingCheck Function: This method ensures that transitions in a state

```
1  void transitionMappingCheck( QClass appOper, QClass conOper,
       MappingModel mapping) {
2    retrieve state machine model for application operation into appModel
3    retrieve state machine model for context operation into conModel
4    get list of transitions in application state machine model
5    for (each transition in application state machine model)
6    begin
7      get mapping for this transition
8      if (mapping not exists)
9      begin
10       throw error and return
11     end
12   end
13   get list of transitions in context state machine model
14   for (each transition in context application model)
15   begin
16     get mapping for this transition
17     if (mapping not eixsts)
18     begin
19       throw error and return
20     end
21   end
22   mapping exists for all states in both source and target state machine
       model
23 }
```

Listing 6.8: Transition Mapping Check Function in QML/CS Validator

machine model for application model and context model can be mapped to each other. It checks that each transition in the application model is mapped to at least one transition in the context model; therefore it satisfies the second condition discussed in Section 5.3. For this it retrieves the application and context model for the operations and then runs a loop on transitions of application model. It finds the first transition mapping in which the set of source transition contains the name of the state in the application model. It then checks that the target transition of the transition mapping exists in the context model. This will check each transition in application model and check in context model if there is at least one transition that is mapped as can be seen in listing 6.8.

transitionConsistencyCheck Function: This function, as can be seen in listing 6.9, ensures the consistency of the mapping between transitions of both application and context operations. It checks that not only that each state in the application model is mapped to at least one state in the context model but the transition between states in the application model should also map to the transition between target states of

the context model mapped in the first step; therefore it satisfies the third condition discussed in Section 5.3. It starts with getting transitions for state machine model of the application model. Then it checks target state for each transition and gets a list of all target states where source state matches with the source name of transition from application model. It then checks in the context model if there is any transition that has same source state and target state and also that a mapping exists between them. It ensures that a transition is mapped only if there is mapping defined and the mapping is consistent. There are other functions like *checkInContextModelStatement* and *checkApplicationModelStatement* to check if a model has associated XMI resource file and is compatible with the model being checked. In addition, they ensures that the XMI file for this context or application model is valid and can be used for checking of states and transitions and their consistency.

6.5 Integrating OCL into QML/CS Grammar

As discussed in Subsection 3.3.7, we extended core OCL grammar [101], especially *FeatureCallExpression* to support the requirements of QML/CS language including *MeasurementCall*, *capacityLimitCall*, *resourceServiceCall* and *helperVariableCall*. The challenge of integrating QML/CS with OCL is concerned with the ability of using OCL expressions within the declaration of our language without using any OCL specific editor. Two mechanisms of OCL integration were considered by either implementing a layer between QML/CS and OCL or embedding raw OCL grammar in QML/CS grammar. Implementing a separate integration layer, as one of the extension and integration mechanism discussed in [5], one of the techniques to integrate with OCL is to restructure the core design of the OCL itself so that languages like QML/CS can integrate with the relevant module like Expressions without worrying about how it works with core grammar modules. But this means that we will have to develop an extension module just for the sake of QML/CS integration and that is not in scope of the thesis, which intends to implement the QML/CS language rather than focusing on extending OCL core architecture. It will make QML/CS specification depending on the extension module or library and any changes in that library would force changes in QML/CS implementation to keep working. Also maintaining the link as well as integration details would complicate the use of QML/CS because of this dependence.

We follow a simpler integration approach to embed the OCL grammar into QML/CS grammar so that the changes made in OCL for QML/CS are independent of any existing extension mechanisms to avoid supporting or maintaining changes for those mechanisms. The option of embedding the OCL grammar into QML/CS grammar saves from all the integration effort and still makes all the OCL features available

```
1  void transitionConsistencyCheck( QClass appOper,  QClass conOper,
       MappingModel mapping) {
2    retrieve state machine model of application operation into appModel
3    retrieve state machine model of context operation into conModel
4    get list of transitions from appModel into _transitions
5    for each transition in _transitions
6    begin
7      for each target for this transition
8      begin
9        get source for this target
10       if (source and target do not map)
11       begin
12         throw error and return
13       end
14     end
15   end
16   get list of transitions for conModel into _conTransitions
17   for (each context transition in _conTransitions)
18   begin
19     for each target in context transition
20     begin
21       if (source and target do not map)
22       begin
23         throw error and return
24       end
25     end
26   end
27   source and targets match for both application and context state
       machine model
28   transitions are consistent
29 }
```

Listing 6.9: Transition Consistency Check Function in QML/CS Validator

```
1 MeasurementDeclaration :
2 ..........
3   'declare' 'measurement' Type name = MeasurementID '('
4 ..........
5   (ownedConstraint+=specConstraints)*
6 '}';
```

Listing 6.10: Grammar snippet for the Measurement in Xtext

to be used directly in the QML/CS specification. This technique allows extension of OCL that specifically supports its working with QML/CS and can be maintained as part of the core QML/CS grammar. It means that QML/CS does not need to maintain compatibility with newer versions of OCL as long as it does not need those features to support specification of NFPs. We decided to use second option because it allows us more flexibility and control on customisation of OCL grammar as well as integration with QML/CS grammar. For example, the line 5 of listing 6.10, shows *(ownedConstraint+=specConstraints)** grammar part, that connect to a grammar expression specified for linking the QML/CS grammar to OCL grammar. The grammar feature *specConstraints* mentions the syntax rules on how the QML/CS can specify an OCL expression and uses the pre-defined OCL grammar features that we embedded in QML/CS grammar. To indicate OCL grammar features in QML/CS, we use the *variableCallExpression spec* so that the expression after this can be evaluated for a valid constraint on the measurement and specify meaning of the measurement. Once we have linked QML/CS with OCL grammar, new grammar rule named *MeasurementExpr* is extended to be part of OCL grammar expression as can be seen in line 10 of listing 6.11 that indicates the OCL expression can be *MeasurementExpr* as well and it should be validated as an OCL expression. That is how it becomes part of OCL expression and makes it available to be used in QML/CS specification. Within this grammar rule, we can refer back to the measurement declaration, specifically lines 23-24 show how to do that. The complete grammar for this listing example can be found in Appendix B.

```
1 SpecConstraints returns ConstraintCS:
2   stereotype='spec' (specification=SpecificationCS ';') ;
3 SpecificationCS returns ExpSpecificationCS:
4   ownedExpression=ExpCS;
5 ......................
```

```
6  PrefixedExpCS returns ExpCS:
7    ({PrefixExpCS} ownedOperator+=UnaryOperatorCS+ ownedExpression=
       PrimaryExpCS)
8  |    PrimaryExpCS;
9  PrimaryExpCS returns ExpCS:
10   MeasurementExp
11 ......................
12 |    TypeLiteralExpCS({NameExpCS} pathName=PathNameCS(
13   (({IndexExpCS.nameExp=current} '[' firstIndexes+=ExpCS (','
       firstIndexes+=ExpCS)* ']'
14   ('[' secondIndexes+=ExpCS (',' secondIndexes+=ExpCS)* ']')?
15   (atPre?='@' 'pre')?)) |({ConstructorExpCS.nameExp=current} '{'
16   ( ((ownedParts+=ConstructorPartCS (',' ownedParts+=ConstructorPartCS)
       *))?
17   | (value=StringLiteral)) '}')|
18   ( (atPre?='@' 'pre')? ({InvocationExpCS.nameExp=current} '(' (
19   argument+=NavigatingArgCS (argument+=NavigatingCommaArgCS)*
20   (argument+=NavigatingSemiArgCS (argument+=NavigatingCommaArgCS)*
21   (argument+=NavigatingBarArgCS (argument+=NavigatingCommaArgCS)*)?)? ')
       ')?)));
22 MeasurementExp:
23   (measurement = [MeasurementDeclaration]) ( '((' '))' '(('
24   ((argument+=MeasurementExpArgument) (',' (argument+=
       MeasurementExpArgument))) '))');
```

Listing 6.11: Grammar snippet for MeasurementExpr rule in Xtext

6.6 Summary

This chapter presented implementation parts of the research, QML/CS Component Architecture, and the actual implementations of the solutions discussed in chapters 4 and 5. The next chapter will give a semantic translation for quality modeling of component-based systems.

7

A Semantic for QML/CS

In the previous chapter, we have discussed implementation of QML/CS. The lifting approach and model mapping implementation was presented and the integration of OCL expression into QML/CS was explained. This chapter will discuss the QML/CS semantics and how they are defined. The semantics are discussed only now because they will be given for the variant of QML/CS finally implemented.

There are many different theories to specify the semantics like static [28], operational [85], axiomatic [91], denotational [92] and translational [27] semantics that can be followed to provide semantics for languages like QML/CS. The static semantics, as the name indicates, has no impact on the dynamic behaviour and focus only on compile-time type checking and validating the declaration so that declarations are resolved based on expected syntax.

The operational semantics represent more about the operational environment in which the program will run so information about c, memory, registers and the way resource are available in an operational environment are specified. A virtual machine is generally needed to be to use operational semantics because it has complete information about operational situation of any system.

The axiomatic semantics, with its original purpose of formal verification of the programs, provides rules in the form of axioms that control the transformation of one expression to another expression and also controls the execution and validation of those transformations to verify that one expression has been converted successfully and legally to the target expression.

The denotational semantics is based on recursive function theory and works by providing a function based notation with each program that needs to be checked for the semantics. The function can call another function if the program being checked for meaning has inner elements that have different functional added as a notation for it. That is why it works in a recursive function way and keeps going into iteration till

all semantics of the program are defined and validated. The translational semantics theory focuses on taking the transformation on a sentence to sentence basis. It picks up a sentence in the source, checks the common elements that have already been transformed and then converts the remaining to the new language. It is important to mention that translational semantics considers semantic of target language to make sure that semantics of a sentence in source language does not change when transformed into new language and compatible semantics in target language are used to provide the transformation.

We have chosen translational semantics for QML/CS because QML/CS is designed as a set of statements with each statement representing a concept. The suitable transformation will need to consider each statement as a separate entity when converting based on the semantics with linking it to any pre-processed concepts. It provides this facility of taking one statement at a time, just like a sentence, and completes the transformation. In this chapter, we show translational semantic for QML/CS language using model transformation technique that will translate the QML/CS specification to TLA+. The target translation to TLA+ language is chosen because there already exists a TLA+ formalisation in [109]. TLA+ is used for the first time.

7.1 Translational Semantic

As mentioned in previous chapters, QML/CS is a high level specification language for NFPs and it provides easy and understandable specification compared to temporal logic expressions. The comparison of syntax with TLA+ would indicate how much abstraction and formalism has been used in QML/CS without compromising the semantics. The comparison is possible if we can translate QML/CS to TLA+ and then evaluate the usability, readability, understandability and comprehensiveness of QML/CS against TLA+ specification produced in this translation. This evaluation part will be discussed in detail in Chapter 8. A translation is a systematic procedure by which any instance of QML/CS meta-model can be transformed into a well-formed TLA+ specification with the same semantics. This translation would need an understanding of TLA+ syntax and representation of semantics that will be compatible with equivalent QML/CS. It means that the translation process has to consider a mapping strategy so that it is clear that which part of QML/CS specification is being translated to which part of TLA+ specification.

Introduced by Abadi and Lamport [62], TLA+ specification is based on the concept of Modules that represent each specification feature. There is a reasonable similarity between object-oriented concepts and the way TLA+ modules are organised and specified. The modules can be linked with each other through association and derivation

so that relationship between the modules can be defined. Three important keywords of TLA+ are MODULE, EXTENDS and INSTANCE that describe declaration of a module, extending a module from another module in the system and association relationship between modules respectively. These concepts show a similarity with object-oriented concepts of class, inheritance and association relationship between the classes. Another important feature of TLA+ is that modules can have other modules specified inside them, called inner modules, that are private to the parent module and other external modules cannot use or derive from them. Since QML/CS model and specification is based on state-machine model and class-model, it is important that TLA+ has an option to represent the same concepts so that a mapping is established. The remaining chapter will present different parts of TLA+ and how they represent the concepts of QML/CS and the technique that has been used to produce TLA+ specification from QML/CS. In the following sections we explain in details about the key challenges associated with a translation mapping.

7.2 Translational Semantic Challenges

Since everything in TLA+ is represented as modules so the first challenge is to be able to convert QML/CS structures to equivalent TLA+ modules. Both specification languages are independent of each other and the TLA+ module structure was not designed to support any future language like QML/CS. It means the transformation technique will need to understand TLA+ syntax and semantics, identify different parts of the specification and then link with QML/CS structure. Epsilon generation language(EGL) [65] is a model to text transformation tool that helps to address this challenge and generates a sensible TLA+ transformation of QML/CS. EGL is a template driven model transformation that takes source model, a template to transform the source model to target specification. Second challenge is that the specification in TLA+ modelled in one place whereas in QML/CS makes more sense to annotate some of them, for example, in the context model and some of them in the measurement specification. This becomes really tricky because finding same meanings of structures in different languages is not an easy task. So, the challenge is how can write consistent model transformation template that links both definition and produce an equivalent TLA+ specification. The solution is to change the way the measurement specification works by explicitly mention in the measurement specification what happens in each transition of context model.

The Model transformation technique is a way of implementing semantic translation so that the model mapping is effectively used in this translation and the context of the specification is maintained. The structure of QML/CS and TLA+ is different and that is one of the differentiating factor that QML/CS can specify a concept in

a simple manner where as TLA+ uses longer and complicated way to specify the same concept. To ensure that concepts from QML/CS are transformed properly into TLA+, a template is used that matches the syntax of QML/CS with syntax of TLA+ while maintaining the semantics. Second major challenge is to ensure the consistency of transformation so that context of QML/CS features is maintained. To address this issue, rules are specified that help to establish link between QML/CS features and their transformed TLA+ features.

7.3 Translational Semantic via Epsilon

In this section, we discuss the actual process of semantic translation using Epsilon for QML/CS specifications. It requires that we write a template for transformation so that each declaration in QML/CS can be transformed into an equivalent TLA+ specification. Since this transformation is done for the actual application so the names of the source declaration should come from the QML/CS language specification. It means that the language specification should also be loaded into the process so that when accessing the context model, the process can find the names of the specification measurements and use the same in TLA+ to generate a logical mapping. There are many different types of declarations in QML/CS like measurement, service, component, resource, container and the way they are specified so the process has to consider each declaration separately and follow the template to do the transformation. The process ensures that there is no hard coding used to provide a feature in TLA+ and each feature in TLA+ is generated based on the context model loaded so that the process is generic and works on any context model loaded to use for transformation. This makes the process automatic that works on the loaded dependencies and then generate a TLA+ specification. It means that if we have another application specification to be transformed into TLA+ specification then we needs its related context model and the process will generate the TLA+ specification without asking for any changes in the transformation code itself.

The translation process from QML/CS to TLA+ is illustrated in Fig 7.1, the right part of the figure shows the transformation process of a model to text that uses template mechanism to map through source models and produces the target as a text. The left part denotes that EGL template takes context model and the actual QML/CS specification as an input from source and map through them based on pre-defined rules to produces TLA+ specifications. For every declaration in QML/CS specification, an equivalent TLA+ specification is produced using QML/CS Context Model. For example, in the measurement declaration, the type of the formal parameters of the measurement definition identifies which part of the context model will be used.

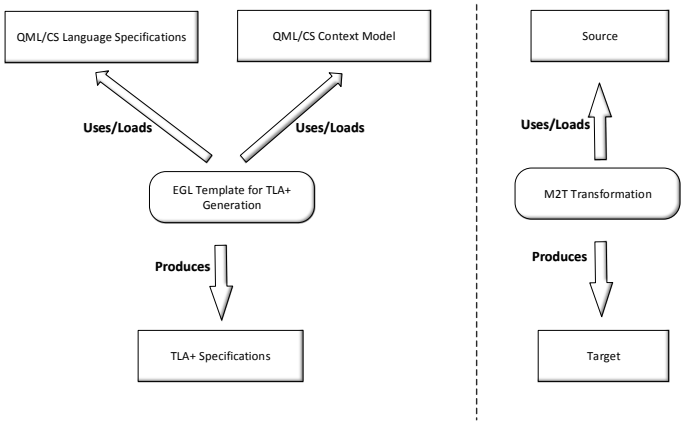

Figure 7.1: An Overview of a Model to Text Transformation via EGL

7.4 EGL Template of Translational Semantic

An EGL template for generation of TLA+ specifications from QML/CS is introduced. It defines a number of rules to invoke multiple templates and generate TLA+ specification from QML/CS specification automatically. For example, listing 7.1 shows main EGL template for the translation of QML/CS specification to TLA+. The listing (line 1 thru 15) shows a number of rules for QML/CS specifications like *Measurement2TLASpec*, *Application2TLASpec*. Each rule consists of two main functions; *template* and *target*. template invokes the specified EGL template like *measurement2TLASpec.egl* (see line 8) and *target* specifies the output file path like *MeasurementsTLASpec.tla* as shown in line 9 of listing 7.1.

```
1  rule ContextModel2TLASpec
2    transform contextModel : ContextModel!ContextModel {
3    template : "ContextModel2TLAPlus.egl"
4    target : "TLA+gen/ContextModelTLASpec.tla"
5  }
6  rule Measurement2TLASpec
7    transform measurement : qmlcs!MeasurementDeclaration {
8    template : "measurement2TLASpec.egl"
9    target : "TLA+gen/MeasurementsTLASpec.tla"
10 }
11 rule Application2TLASpec
12   transform application : qmlcs!ApplicationModelStatement {
13   template : "Application2TLA.egl"
14   target : "TLA+gen/ApplicationTLASpec.tla"
15 }
```

Listing 7.1: EGX snippet for the semantic translation of TLA+ specification rules

The listing 7.2 shows what will be input to the transformation template and it will be converted to TLA+ specification for the measurement. It starts (Lines 1 thru 4) with the definition of measurement *Response Time*, which is defined for single service operation op as a difference between end and start time of the last invocation of that operation.

```
1  in context RT;
2  declare measurement Real response_time (ServiceOperation op){
3  On op.Init update
4        ResponseTime = 0;
5        hadOpCall = FALSE;
6  On op.RequestArrival update
7              start = 0;
8              end = 0;
9  On op.StartRequest update
10             start = now;
11 On op.FinishRequest update
12             end = now;
13             ResponseTime = ResponseTime + end;
14             ResponseTime = ResponseTime - start;
15             hadOpCall = TRUE;
16 }
```

Listing 7.2: The QML/CS specification for *Response Time*

Listing 7.3 shows the TLA+ specification of *response time* module. It is equivalent to QML/CS specification of the *response time* discussed in listing 7.3. Lines 2-9 extend *Real Time* module, make use of the module of a *service* and lines 11-20 shows specifying the difference between end and start time of the last invocation of the single service operation. The variables *start* and *end* represents the start and end time of the operation. The *Spec* in this listing shows the state machine specification with a list of states that are part of state machine model of the measurement context.

Listing 7.4 shows an example pseudo-code snippet of an EGL template for the semantic translation of measurement declaration. It integrates with QML/CS specification and extracts information about the measurement, variables used to handle the measurement, different transitions that compose the measurement and the values assigned to variables at different stages. For now, line 3 is of interest, a loop is defined iterating over all measurements in a system specification where *MeasurementDeclaration* is a list of all measurements defined in the system. Each item in this list represents QML/CS concept of *Measurement*, which has a name that is defined as property

```
1  ---------- MODULE response_time ----------
2  EXTENDS RealTime
3
4  VARIABLES inState, unhandledRequest
5  VARIABLES start, end
6
7  op == INSTANCE RT
8  ---------------------------
9  VARIABLE ResponseTime
10 VARIABLE hadOpCall
11 ---------------------------
12 OnInit == op!Init=>
13                   /\ response_time \in Real
14                   /\ ResponseTime= 0
15                   /\ hadOpCall= FALSE
16
17 OnRequestArrival == op!RequestArrival =>
18                                     /\ start = 0
19                                     /\ end = 0
20
21 OnStartRequest == op!StartRequest =>
22                                     /\ start'= now
23
24 OnFinishRequest == op!FinishRequest =>
25                                   /\ end'= now
26                                   /\ ResponseTime'= ResponseTime + end
27                                   /\ ResponseTime'= ResponseTime - start
28                                   /\ hadOpCall'= TRUE
29 RTSpec == /\ op!RT
30           /\ [][ OnInit/\OnRequestArrival /\ OnStartRequest /\
       OnFinishRequest]_<<ResponseTime, hadOpCall>>
31 ===========================
```

Listing 7.3: The TLA+ specification for *Response Time*

name that is being accessed using the iteration object *ms* for each measurement in the list, as shown in line 5. Line 6 shows the loop for each measurement definition where each definition is a combination of variables used to define the measurement and expressions defined as part of measurement specification. Lines 7 to 14 show how to handle multiple occurrences of a variable. It uses a set variable to see if the next variable occurrence should be added to list so that required variables list in TLA+ does not contain any duplicate values. These variables are made part of the output to indicate corresponding measurement variables that will be generated from measurement specifications in QML/CS.

The measurement context *msCxt* has a property *Trans* that has a list of all transitions for this measurement. Line 15 explains how the transitions are retrieved from the source QML/CS specification and then transformed into equivalent TLA+ transitions. It extracts the name of each transition with required prefixes based on if it is first transition or last transition or anywhere between them because TLA+ expects some specific literals like "==" attached based on the sequence of the transitions. Lines 19 and 20 show how it extracts the variables and expressions for each transitions and then converts that to TLA+ representation of variable expressions. Lines 21 thru 31 explain how it checks the type of variable expression so that it can generate an equivalent transformation for the expression. The different possible types are *AttributeCSExp*, *VariableValue*, *NumberLiteralExpCS* and *BooleanLiteralExpCS*. The last transition need to show the output of the measurement as well. Once this information is extracted, the template provides a target representation of TLA+ so that extracted content can be embedded in TLA+ specification. Then the template is parsed and executed by EGL in Eclipse IDE environment to generate TLA+ output.

7.5 Code Generator Testing

There were two considerations made to test the code generation. First one was to ensure that the TLA+ code generated has the correct syntax as per TLA+ standard. Once the syntax is validated, second consideration was to make sure that the code represents QML/CS concepts completely. The inspiration was also taken from the sample TLA+ representations presented in [109] and check that the generated code confirms to that format. It will ensure that the code not only confirms to TLA+ standard but it also builds on what is already presented in [109]. Two strategies of testing are followed to validate the code generator; as Unit Testing and System Testing. Unit testing is needed so that transformation of each QML/CS concept can be validated individually before it is tested combined with other features of QML/CS. The Unit tests were conducted in following steps:

```
void TransformMeasurementDeclaration(){
  variableSet := initialise with empty list
  for each measurement declaration in the QML/CS specification
    print measurement model name
    assign measurement declaration into ms
    for each measurement definition in the current declaration
      for each variable in measurement variables list
        for each expression in measurement definition
          if class of expression = "AttributeExpCS" then
            if not variableSet contains variable representing the
    expression then
              add variable to variableSet
    if not count of variableSet elements is greater than 0 then
      return  error
    print list of variables in variableSet
    for each measurement transition
      print transition name
      for each parameter in measurement declaration
        print parameter name
        for each variable in the measurement declaration
          for each expression in variable declaration
            if expression class = "AttributeExpCS" then
              if name of expression variable is first one then
                print the name only
              for each expression reference in expression references
                if expression is "ownedOperator" then
                  for each operator in the operators list
                    if class of operator is "VariableValue" then
                      print variable operator name
                    else if class of operator is "NumberLiteralExpCS"
    then
                      print numeric operator name
                    else if class of operator is "BooleanLiteralExpCS"
    then
                      print boolean operator name
    for each parameter in measurement
      if type of parameter = "ServiceOperation" then
        print parameter name with "Service" post fix
      if type of parameter = "ComponentOperation" then
        print parameter name with "Component" post fix
    for each transition in measurement declaration
      if it is last transition
        print transition name with "_measurementname" post fix
      else
        print transition name only
}
```

Listing 7.4: Pseudo-code snippet with typed iterator of Measurement Declaration

- A transformation template was written to work as transformer between the QML/CS feature and equivalent TLA+ representation.
- The template takes two inputs; the model and the QML/CS code.
- The EGL template generates the TLA+ code.
- The code is checked in a tool called *The TLA Toolbox*, which confirms that syntax is according to the grammar of TLA+ language.
- The code is then checked manually with the samples provided in [109] to ensure that the structure of both codes is similar.

Once the individual features were tested and their syntax was validated to be representing the equivalent QML/CS features, the sub templates were combined into a major container template to check that the TLA+ generated for the whole application in question is valid. The main template will load the templates along with source model to use for transformation. Each template will be launched to complete its part of transformation and then all the transformations are joined together to make one representation in TLA+. This transformation is then validated in the *TLA+ Toolbox* and confirmed that all the concepts are represented as per requirement.

7.6 Summary

In this chapter, we have shown that QML/CS language can be transformed into TLA+ specification while maintaining the semantics of QML/CS. Also the challenges faced in transformation were addressed and a reasonable transformation is produced. The process of transformation is also explained and the templates needed for transformation are elaborated to exhibit how structures of QML/CS are mapped to equivalent TLA+ structures. It gives an indication for compatibility of QML/CS with the framework it is based on and also its ability to be transformed to other specifications based on templates. It also shows that a high level specification can be produced in QML/CS and then automatic transformation can be used to convert that to any desirable specification format. In the next chapter we will show the evaluation of our QML/CS language that provides supporting evidence that it can give a practically usable specification for NFPs of component-based systems.

8 Evaluation

In the previous chapter, we have discussed a semantic translation for QML/CS language using model transformation technique that translates the QML/CS specification to TLA+. We have seen that the specification of QML/CS can be transformed into TLA+ spec. This chapter describes the experimental evaluation of QML/CS. The usage of the prototype from the perspective of QML/CS is explained as a case study.

8.1 A Case Study

This chapter aims to evaluate the ability of the proposed QML/CS language to define Non-Functional Properties (NFPs) of component-based systems. The evaluation targets an industrial application for which NFPs can be defined using QML/CS. A case study [11] was selected to show the applicability of QML/CS in specifying to this application. We demonstrate how to apply QML/CS to this application. A complete set of QML/CS models of the application is provided along with discussing on how these models would be designed and developed. We would later present the TLA+ specifications generated by the code generator.

The web audio store application is used to demonstrate the QML/CS claims of providing NFPs specification of a complex component-based system. We explore architecture of the web audio store which is an on-line audio streaming website. The user can upload and stream different available audio files in the database. This case study only focuses on the NFPs of the system and all the aspects of software architecture for functional properties is out of scope for this evaluation. In commercial applications (e.g. web audio store), the definition of NFPs becomes as important as defining functional requirements. There are not many solutions available in development community, which provide means to define these NFPs of a component-based system generically.

8.1.1 Overview of the Case Study

In order to support our claim that we provide a practically usable generic language for defining NFPs of component-based system, we start with presenting the architecture of selected web audio store application [11] as shown in Fig 8.1. This example is suitable for our evaluation because it represents an application which is structured based on component-based system architecture. This case study attempts to solve the problem that there does not exist any practically usable and generic language for specification of NFPs. The details of the case study are elaborated in the following sections. We specify the web audio application using QML/CS language and defined a number of measurements generically and have applied those measurements to concrete operations of the audio system. This successful specification of measurements on a practical component-based system justifies the claim of practical usability of QML/CS. The case study was selected around the objective that it can help answering following questions:

- Can the QML/CS language be successfully applied to specify NFPs of a real system?
- Can we derive from this case study that the QML/CS language is a practically usable language for modelling NFPs generically?
- What challenges are faced in applying QML/CS language? this is critical in order to identify gaps when applying QML/CS to model NFPs.

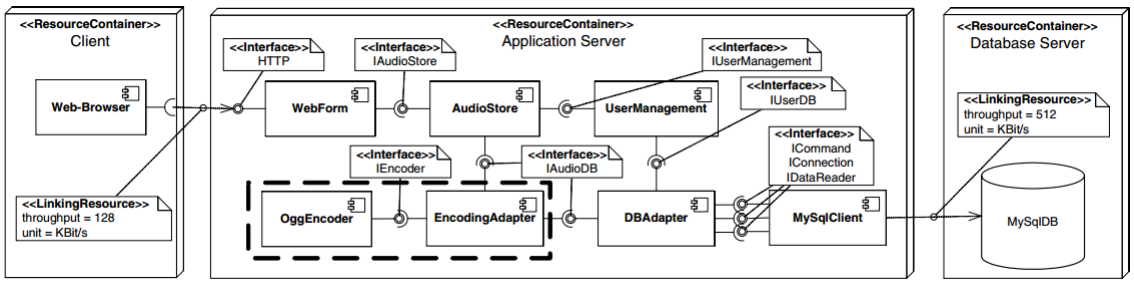

Figure 8.1: Web audio store architecture derived from [11]

We present the important specification parts of the application case study using QML/CS and the remaining specification can be found in Appendix A.1. We also present TLA+ translation for the selected specification elements and remaining TLA+ specification can be found in Appendix A.2. The specification are being shown below in small listings with brief explanation.

8.2 QML/CS Specification

The QML/CS specification for the selected application includes a number of declarations for a measurement, service/component, resource, container and system to specify them along with referencing the context, application and mapping models. QML/CS then loads the required models (e.g, context model, application model and resource model). These models are fully developed and instantiated using QML/CS. The QML/CS users need to instantiate these models while writing the specification of system. Once the models are loaded, the user can use the desirable models and its elements to complete the specification of the system. Each declaration of QML/CS involves writing a specification for all those elements including OCL expression if applicable.

We discuss a number of models of the presented case study including the context models (used by measurements), application models (used by component to declare the applications) and resource models (used to write the specification of abstract and concrete resources). QML/CS declarations can be used by various users like Measurement designer, Component Designer and Platform Designer during the complete development cycle of the system. The forth main prospectives of QML/CS language concerning the different usages of the language are presented in the following Subsections.

8.2.1 Measurement Designer's Perspective

The measurement designer is one of the roles for users to use the QML/CS specification for formal specification of non-functional properties. Its perspective covers many different expectations from the language and are listed below

1. Analysis requirements: Understanding of the requirement specification to be able to know what non-functional perspectives exist in the system.

2. Identify NFPs: Identify specific non-functional properties that are required to be met by the target system. Identification of the components and services with which these NFPs are attached.

3. Choose the appropriate context model so that measurement can be defined without dependency on the application model. Identify which context model is suitable for each measurement.

4. Define measurements: provide a formal definition of identified measurements and their constraints.

5. Add measurement definitions to repository: Add the specification to a repository so that application designer can reference them in complete specification.

This subsection presents the context model and the specification of the application from Measurement Designer's perspective.

8.2.1.1 Measurement's Context Model

Context models are used to specify measurements independently of concrete applications, therefore, measurement specifications are based on the context model specifications of component or service as discussed in Section 3.3.1. Each measurement specification should reference a context model and in this case study we reference context models as can bee seen in figures 8.2 and 8.3 . Context models *RT* and *ET* are referenced by keyword 'in context', followed by the name of the model. Each model consists of various classes and associations between them as follows:

1. Parameter's type of measurement: it is *Service Operation* in the context model (RT) and *Component Operation* in the context model (ET).

2. State Machine Model: it defines the behaviour of measurement parameter's type. It consists of a number of states and transitions (e.g. idle, RequestAvailable and HandlingRequest) and transitions (e.g. RequestArrival, StartRequest and FinishRequest) in context model (RT) whereas in context model (ET), the state machine model includes one additional state named *Blocked* and two transitions named *SwitchToOther* and *SwitchBack*.

3. Each parameter type of measurement has a property named State machine model: this is to associate the parameter type of measurement with its own state machine model in order to define its behaviour.

4. The measurement designer loads parameter type of the measurement and transitions of relevant state machine model are needed to complete the definition of a measurement.

8.2.1.2 Measurement Specification

Two measurements *response time* and *execution time* are defined for the Web Audio Store system as shown in listings 8.1 and 8.2. They indicate that both *response time* and *execution time* measurements reference their context models *RT* and *ET*, and show the time events that should be considered in their calculation.

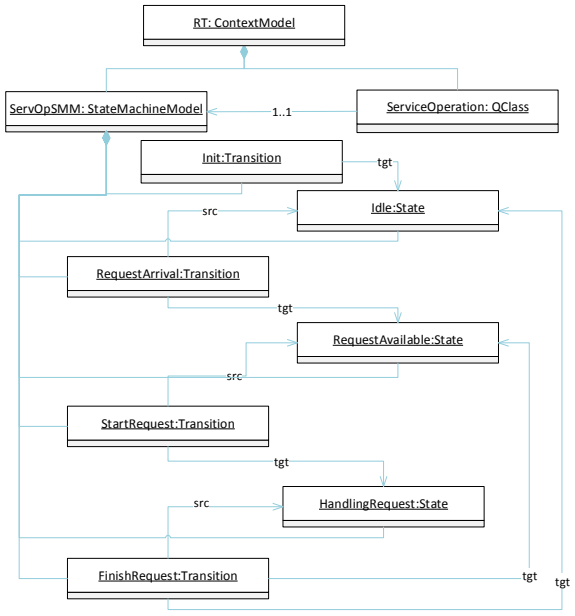

Figure 8.2: Context Model for Response Time (RT).

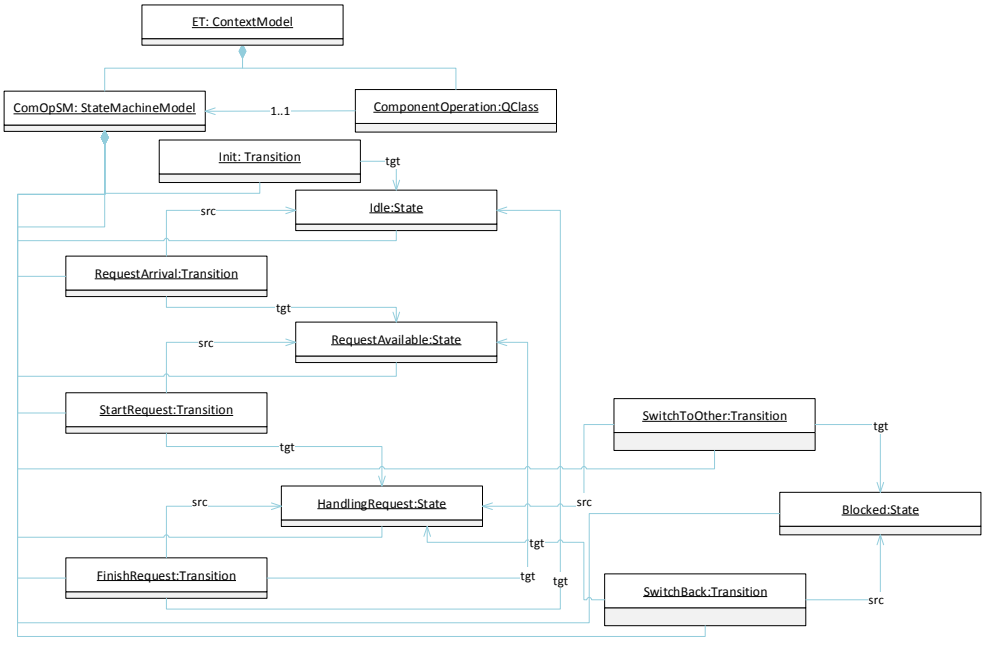

Figure 8.3: Context Model for Execution Time (ET).

The listing 8.1 shows the QML/CS specification for measurement *response time*. Line 1 starts with referencing a context model named *RT* for the *response time*. Line 2 shows how the *response time* is expressed, that is defined for single service operation *op* as a difference between end and start time of the last invocation of that operation. The context model *RT* is referenced to allow a measurement designer to load the type of *response time* measurement's parameter *Service Operation* and some transitions of state machine model that are of interest to this particular measurement as shown in lines 3 thru 11 along with specifying some values on those transitions based on on the measurement definition.

```
1  in context RT;
2  declare measurement Real response_time (ServiceOperation op){
3  On op.Init update
4          ResponseTime = 0;
5          hadOpCall = FALSE;
6  On op.RequestArrival update
7                start = 0;
8                end = 0;
9  On op.StartRequest update
10              start = now;
11 On op.FinishRequest update
12              end = now;
13              ResponseTime = ResponseTime + end;
14              ResponseTime = ResponseTime - start;
15              hadOpCall = TRUE;
16 }
```

Listing 8.1: The QML/CS specification for *response time*

The listing 8.2 shows the QML/CS specification for measurement *Execution Time*. Line 1 starts with referencing a context model named *ET* for the *execution time* measurement. Line 2 shows how this measurement is expressed, that is defined for single component operation *op1* as a difference between end and start time along with subtracting blocked time, if any, of the last invocation of that operation. A context model *ET* is referenced to allow a measurement designer loads the type of *execution time* measurement's parameter *Component Operation* and some transitions of state machine model that are of interest to this particular measurement as can be seen in lines 3 thru 20 along with specifying some values on those transitions based on on the measurement definition.

```
1  in context ET;
2  declare measurement Real execution_time (ComponentOperation op1){
3  On op1.Init update
4          AccExec = 0;
5          SegStart = 0;
```

```
6         Execution_time = 0;
7         hadOp1Call = FALSE;
8   On op1.StartRequest update
9             SegStart = now;
10            AccExec = 0;
11  On op1.FinishRequest update
12            Execution_time = AccExec + now - SegStart;
13            hadOp1Call = TRUE;
14  On op1.SwitchToOther update
15            AccExec = AccExec + now - SegStart;
16  On op1.SwitchBack update
17            SegStart = now;
18  }
```

Listing 8.2: The QML/CS specification for *execution time*

8.2.2 Application Designer's Perspective

In the previous subsection, we discussed the measurement specifications used in our case study concerning the measurement designer's prospective. In this subsection, an application designer's perspective is presented that covers following in using the language.

1. Analyse measurements from the repository: The identified measurements are checked if they cover all the NFPs needed to be implemented for the system.

2. Determine what operations to be specified in the specification for each service or component.

3. Specify services or components: Write formal specification of each service or component of the target system using the identified QML/CS features. Writing the mapping models needed for a service or component.

8.2.2.1 Application Model

This subsection presents the application model and the specification of Application Designer's perspective. Application models are used to specify concrete applications, therefore, application specifications are based on the application model of either a service or component as discussed in Subsection 3.3.3. In the application declaration for a service or component, the application model is referenced via the keyword *application* followed by name of that application (e.g. application ComponentNameModel).

This is to refer to a location where the 'application model' can be found. The application model consists of an operation type and an associated *state machine model*. Each operation type has a property named State machine model to reference its own state machine model, which defines its behaviour. The State Machine Model consist of a number of states and transitions (e.g. OpNameidle, OpNameRequestAvailable, OpNameHandlingRequest) and transitions (e.g. OpNameRequestArrival, OpNameStartRequest and OpNameFinishRequest). Operation Type (e.g. Component Operation) is then loaded into the service or component declaration to define its type operation and behaviour. There are a number of application models used in the case study as specified for components and a service. We will not discuss them in details and only focus on two application models as shown in figures 8.4 and 8.5, and the remaining models can be found in Appendix A.

Fig 8.4 shows the Audio Rental model that contains *Service Operation* type and a state machine model. This state machine model has a number of states (e.g. IdleRentAudio, ReceivedRentAudio) and transitions (e.g. RARequestArrival, RARentAudioStartRequest) that define the behaviour of the operation *rentAudio()*. Fig 8.5 shows Web Form Model that consists of *Component Operation* type and a state machine model. This state machine model has a number of states (e.g. IdleUploadFile, ReceivedUploadFile) and transitions (UFRequestArrival, UFStartRequest) that defines the behaviour of the operation *uploadFile()*.

8.2.2.2 Application Specification

In this subsection, we provide the application specifications for *Web Audio Store* and remaining application specification can be found in Appendix A.1. Each specification references the application model (of a service/component) via 'application' key word followed the name of the application model (e.g. application ComponentNameModel).

The listing 8.3 shows *Web Audio Store* application that has a number of declarations to specify its services/components, resource, container and system. Line 1 starts with referencing the application model named *AudioRentalModel* for *AudioRental* Service. This model consists of *Service Operation* type and its own state machine model to define its behaviour as shown in Fig 8.4. Line 2 demonstrates the declaration of *AudioRental* Service followed by line 3 which begins with the key word *provides* and *rentAudio* operation including its type and returned type to describe the service's interface that is part of the non-functional properties specification. Line 4 in the listing starts with the key word *always* that uses a syntax based on Object Constraint Language (OCL) to specify the meaning of the measurement. The measurementCall *response time* of the *rentAudio()* is constrained to 20 milliseconds along with the loading mapping model named *RAOp2RtMapping*.

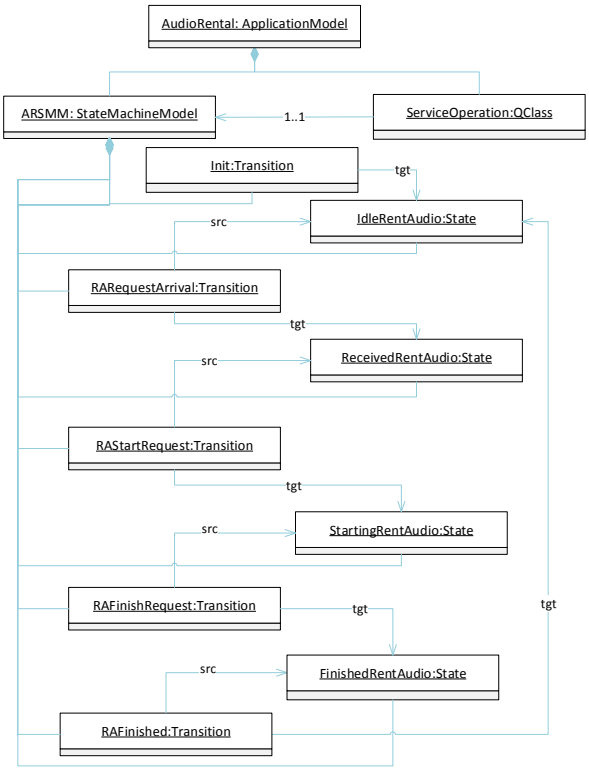

Figure 8.4: Application Model for Audio Rental Service.

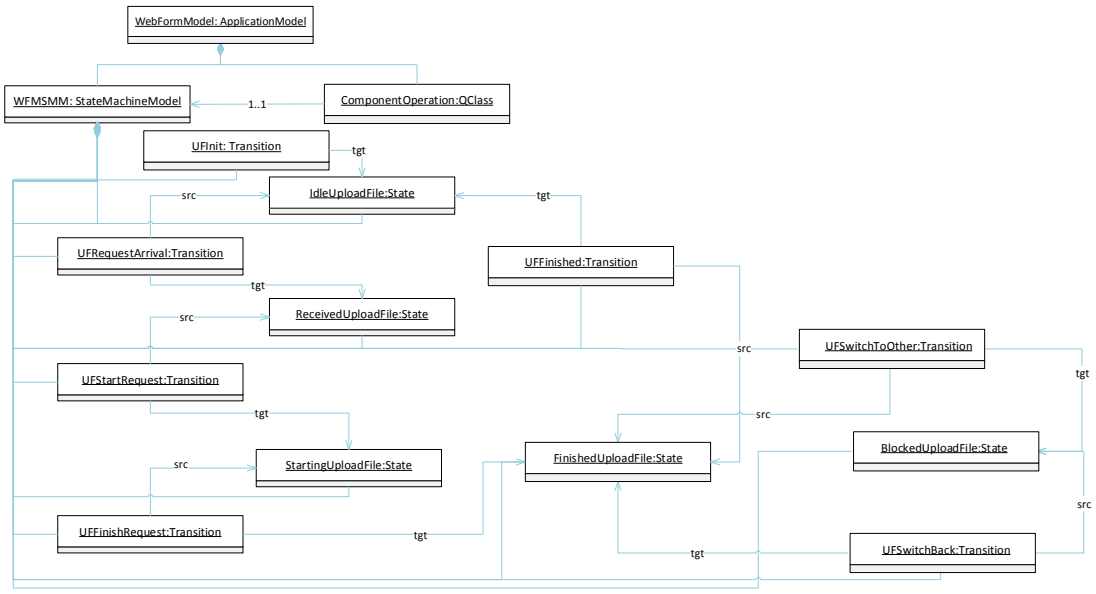

Figure 8.5: Application Model for Web Form Component.

To apply a measurement to a concrete application a mapping model must be referenced in measurement's argument providing the name of the mapping model. This loads a mapping model so that it can be used by *QML/CS Validator* to validate the target operations. A mapping model clearly describes structure of the mapping to link different features of both application model of operation and context model of measurement. Each mapping model has a number of mapping strategies that help in mapping the application to context model. These includes *ClassMapping*, *StateMachineModelMapping*, *StateMapping* and *TransitionMapping* for establishing a link between application model and context model.

Validating the *rentAudio()* operation is achieved by loading the *AudioRentalModel* and *RT* context model so that the mapping between these models can be established. This mapping is established in the mapping model named *RAOp2RtMapping* as shown in Fig 8.6. The *ServClassMapping* controls the mapping between source and target *Service Operation* classes in application model and context model respectively. It is important to note that each *ClassMapping* should have a reference to *StateMachineModelMapping* to verify that a particular class belongs to its own sate machine model. *StateMachineModelMapping* controls the mapping between source and target state machine model named *SSMMStateMachineModelMapping* that contains a list of all the states and transitions that belong to the classes in the *AudioRentalModel*

and *RT* context model. The *StateMapping* and *TransitionMapping* control mapping of each state and transition in state machine model of the *AudioRentalModel* to a state and transition in state machine model of the *RT* context model as required by the *StateMachineModelMapping*.

As discussed in Chapter 5, we show application of the three conditions to examples of the mapping models in our case study. Fig 8.6 shows set of states and transitions; and the mapping instances that connect the states and transitions between application and context models. The equation one requires that for each states in state machine model of *AudioRental* application model, there is at least one state in the *RT* context model to which it can be mapped. We can see in Fig 8.6 that each of the states like *IdleRentAudio*, *ReceivedRentAudio* and *StartingRentAudio* are linked to relevant states in the *RT* context model via state mapping objects like *IdleSM* and *RequestAvailSM*. Fig 8.6 also shows interesting state mappings happened, where a sequence of states (e.g. StartingRentAudio and FinishedRentAudio) in the *AudioRental* application model mapped to one state (e.g. HandlingRequest) in the *RT* context model.

The second equation requires that for each transition in state machine model of *AudioRental* application model there is at least one transition in the *RT* context model to which it can be mapped. It is also evident from the figure that transitions like *RARequestArrival*, *RAStartRequest* and *RAFinishRequest* in *AudioRental* application model are mapped to transitions in the *RT* context model via transition mapping objects like *RequestArrTM*, *StartRequTM* and *FinishRequTM*. Fig 8.6 also shows interesting transition mappings occurred, where a sequence of transitions (e.g. RAStartRequest and RAFinishedRequest) in the *AudioRental* application model mapped to one *SelfTransition* in the *RT* context model. The *SelfTransition* helps with building a simulation relationship between the states when the source and target are same. Moreover, the third equation is considered where mapping of a each pair of states in the *AudioRental* application model to at least one pair in the *RT* context model are evaluated. This is to ensure that mapping within the states and transitions of *AudioRental* application model and *RT* context model is checked to be consistent.

Lines 6 thru 10 in the listing 8.3 show specification of a component *WebForm*. It begins with referencing the application model *WebFormModel* for *Web Form* Component. This model consists of *Component Operation* type and its own state machine model to define its behaviour as shown in Fig 8.5. Line 7 demonstrates the declaration of *WebForm* Component followed by line 8 starting with the key word *provides* and *uploadFile* operation including its type and returned type to show the Component's interface that is part of the non-functional properties specification. Line 9 starts with the key word *always* and followed by the measurementCall *execution time* of the *uploadFile()* that is constrained to 20 milliseconds along with the loading mapping

```
1  application AudioRentalModel;
2  declare Service AudioRental{
3    provides Operation int rentAudio();
4    always response_time(rentAudio by RAOp2RtMapping.Mapping1)<20;
5  }
6  application WebFormModel;
7  declare Component WebForm{
8    provides ComponentOperation int uploadFile();
9    always execution_time(uploadFile by UFOp2EtMapping.Mapping1) <20;
10 }
11 application AudioStoreModel;
12 declare Component AudioStore {
13   provides ComponentOperation int subscribe();
14   always execution_time(subscribe by SOp2EtMapping.Mapping1) <20;
15 }
16 application DBAdapterModel;
17 declare Component DBAdapter {
18   provides ComponentOperation int read();
19   always execution_time(read by ROp2EtMapping.Mapping1) <40;
20 }
21 application UserManagementModel;
22 declare Component UserManagement {
23   provides ComponentOperation int authenticateUser();
24   always execution_time(authenticateUser by AUOp2EtMapping.Mapping1) <=
       30;
25 }
26 application OggEncoderModel;
27 declare Component OggEncoder{
28   provides ComponentOperation int encodeAudioData();
29   always execution_time(encodeAudioData by EAOp2EtMapping.Mapping1) <=
       30;
30 }
31 application EncodingAdapterModel;
32 declare Component EncodingAdapter {
33   provides ComponentOperation int processEncoding();
34   always execution_time(processEncoding by PEOp2EtMapping.Mapping1) <=
       30;
35 }
36 application MySqlClientModel;
37 declare Component MySqlClient {
38   provides ComponentOperation int authenticateUser();
39   provides ComponentOperation int storeAudioFile();
40   provides ComponentOperation int getUserAudioSubscriptions();
41   provides ComponentOperation int loadAudioFile();
42   always execution_time(storeAudioFile by SAOp2EtMapping.Mapping1) <=
       30;
43 }
```

Listing 8.3: Web Audio Store Application Components Specified via QML/CS

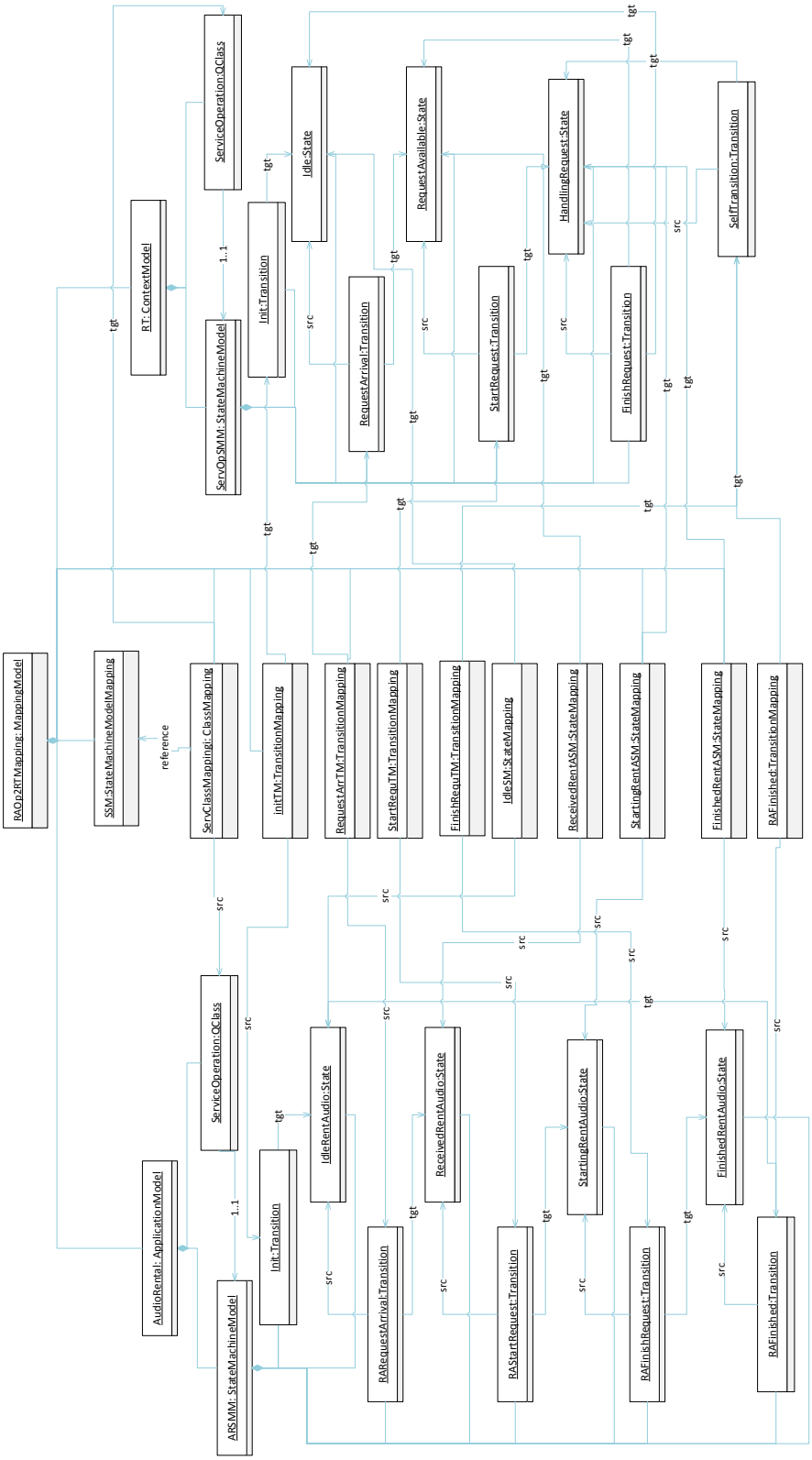

Figure 8.6: *RAOp2RtMappingModel* for Validating *RentalAudio* Operation.

model namedUFOp2EtMapping. The mapping model *UFOp2EtMapping* is used to validate the behaviour of the *uploadFile()*.

Lines 11 thru 15 in the listing 8.3 show specification of a component *AudioStore*. It begins with the reference to the application model *AudioStoreModel* for *AudioStore* Component. This model consists of *Component Operation* type and its own state machine model to define its behaviour. Line 12 demonstrates the declaration of *AudioStore* Component followed by line 13 starts by the key word *provides* and *subscribe* operation including its type and returned type. Line 14 starts with the key word *always* and followed by the measurementCall *execution time* of the *subscribe()* that is constrained to 20 milliseconds along with the loading mapping model *SOp2EtMapping*. The mapping model *SOp2EtMapping* is used to validate the behaviour of the *subscribe()*.

Lines 16 thru 20 in the listing 8.3 show specification of a component *DBAdapter*. It begins with the reference to the application model *DBAdapterModel* for *DBAdapter* Component. This model consists of *Component Operation* type and its own state machine model to define its behaviour. Line 17 demonstrates the declaration of *DBAdapter* Component followed by line 18 which starts with the key word *provides* and *read* operation including its type and returned type. Line 19 starts with the key word *always* followed by the measurementCall *execution time* of the *read()* that is constrained to 40 milliseconds along with the loading mapping model *ROp2EtMapping*. The mapping model *ROp2EtMapping* is used to validate the behaviour of the *read()*.

Lines 21 to 25 show specification of a component *UserManagement*. It begins with the reference to the application model *UserManagementModel* for *UserManagment* Component. This model consists of *Component Operation* type and its own state machine model to define its behaviour. Line 22 demonstrates the declaration of *UserManagement* Component followed by line 23 which starts with the key word *provides* and *authenticateUser* operation including its type and returned type. Line 24 starts with the key word *always* followed by the measurementCall *execution time* of the *authenticateUser()* that is constrained to 30 milliseconds along with the loading mapping model *AUOp2EtMapping*. The mapping model *AUOp2EtMapping* is used to validate the behaviour of the *authenticateUser()*.

Lines 26 to 30 show specification of a component *OggEncoder*. It begins with the reference to the application model *OggEncoderModel* for *OggEncoder* Component. This model consists of *Component Operation* type and its own state machine model to define its behaviour. Line 27 demonstrates the declaration of *OggEncoder* Component followed by line 28 which starts with the key word *provides* and *encodeAudioData* operation including its type and returned type. Line 29 starts with the key

word *always*, followed by the measurementCall *execution time* of the *encodeAudioData()* that is constrained to 30 milliseconds along with the loading mapping model *EAOp2EtMapping*. The mapping model *EAOp2EtMapping* is used to validate the behaviour of the *encodeAudioData()*.

Lines 31 to 35 show specification of a component *EncodingAdapter*. It begins with the reference to the application model *EncodingAdapterModel* for *EncodingAdapter* Component. This model consists of *Component Operation* type and its own state machine model to define its behaviour. Line 32 demonstrates the declaration of *EncodingAdapter* Component followed by line 33 which starts with the key word *provides* and followed by *processEncoding* operation including its type and returned type. Line 34 starts with the key word *always*, followed by the measurementCall *execution time* of the *processEncoding()* that is constrained to 30 milliseconds along with the loading mapping model *PEOp2EtMapping*. The mapping model *PEOp2EtMapping* is used to validate the behaviour of the *processEncoding()*.

Lines 36 thru 43 show specification of a component *MySqlClient*. It begins with the reference to the application model *MySqlClientModel* for *MySqlClient* Component. This model consists of *Component Operation* type and its own state machine model to define its behaviour. Line 37 demonstrates the declaration of *MySqlClient* Component followed by lines 38 to 41 starting with the key word *provides* and the operations *authenticateUser*, *storeAudioFile()*, *getUserAudioSubscriptions()*, *loadAudioFile()* including their types and returned types to show the Component's interface that are part of the non-functional properties specification. Line 43 starts with the key word *always* and followed by the measurementCall *execution time* of the *storeAudioFile()* that is constrained to 30 milliseconds along with the loading mapping model *SAOp2EtMapping*. The mapping model *SAOp2EtMapping* is used to validate the behaviour of the *storeAudioFile()*.

8.2.3 Platform Designer's Perspective

This subsection presents the resource model, the specification of resource and container specified by platform designer. The platform designer has to consider specification of platform and the operational environment so that need and expectations from required resources can be specified. This process will define the relationship of measurements with their corresponding resources in the abstraction of a *container* concept so that capacity and availability of the resource can be specified against demand to perform tasks. Platform designer writes specification about resources, containers and application context models attached with the measurement so that a complete formal specification is available. The specification for a resource and container is presented

in next subsections.

8.2.3.1 Resource's Model and Specification

While defining the specification for resource (e.g. DB), the platform designer references to the resource model via the key word 'in context', followed by the name of the resource model. Resource Type demand (e.g. Query) highlights the type of expression that the individual resource demands. It define an individual query demand. State-Machine model of the resource defines behaviour of the resource and constantly assigns query to the the database resource. It has states and transitions (e.g. idle, RequestAvailable, HandlingRequest) and transitions (e.g. RequestArrival, StartRequest and FinishRequest) as shown in Fig 8.7. Users can specify a number of events on those transitions (e.g. assignedTo, QueryCount) including resource Function (e.g. TimeTaken). It allocates the time taken by *DB* to perform each query. Users can specify capacity limit of resource to state that its conditions are satisfied so that the resource can provide its service.

The *MyDBModel* references a state-machine model of a process that is submitting queries to the database resource so that they can be scheduled for execution. *MyDBModel* provides a function called *TimeTaken* that measures the time taken to execute a query and a collection *scheduledQueries* of (*id*, *demand*) that represents the queries being executed in the database. The type Query defines how a query information should be structured. It has a field called *met* that represents maximum execution time allowed for a query when the resource database is being used fully. The specification in listing 8.4 indicates that all queries in the demand set are being executed on the *DB* and they meet the constraint of maximum execution time for each of them. Lines 1 to 9 show how an accumulative constraint is defined and the way it depends on each query task. Using the model *MyDBModel*, the resource *DB* can be defined as follows:

```
1  in context MyDBModel
2  declare abstract resource DB {
3    demand Query;
4    service(Set(Query) demand) = always
5      scheduledQueries->collect(q | q.demand)
6      ->includesAll(demand) and scheduledQueries->size() = demand->size()
       and scheduledQueries->forAll(q | TimeTaken(q.id) <= q.demand.mrt);
7
8    always (capacityLimit(demand) => service(demand));
9  }
10
11 declare resource MySQLDB of DB {
12   capacityLimit (Set(Query) demand))
13   = demand->iterate (q: Query;
```

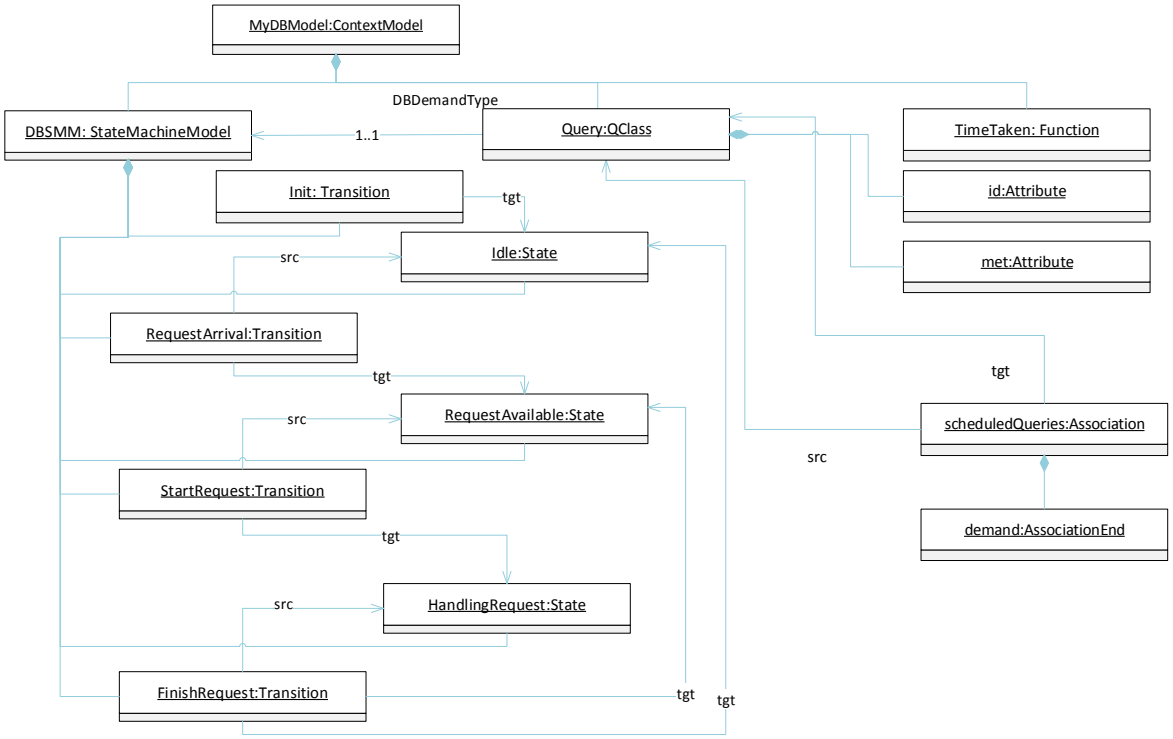

Figure 8.7: Resource Model for Database resource.

```
14      qmet: Real |
15      qmet + q.met) <= demand->size() * 3;
16  }
```

Listing 8.4: Web Audio Store Resource Specification via QML/CS

Lines 11 to 15 in the listing 8.4 show the concrete resource *MySQLDB* specification. Each query limited to max 3 seconds and *qmet* refers to the total maximum execution time. The *capacityLimit* function has to be defined for concrete *DB* that indicates how the capacityLimit is enforced on the resource. It also indicates that capacity is not just about number of queries that can be run but also maximum execution time for all queries should not go beyond three seconds for each query.

8.2.3.2 Container Specification

Listing 8.5 demonstrates an example of a container named *AudioSystemContainer*, that is based on the architecture shown in Fig 8.1. It shows the Audio System container specification, the required part of this specification describes the *MSqlClient*

component and database resource used by the container. The database resource specification in the container has some constrains, by which a capacity requirement for some abstract resource is specified. The last part of the container specification *provides* refers to the services provided by the container. The container strategy for query execution time is provided below based on the previous examples.

```
declare container AudioSystemContainer(ResponseTime: Real) {
  ExecutionTime: Real;
  requires
  Component WebForm {
    provides uploadFile();
  always execution_time(uploadFile) < ExecutionTime;
  };
  Component DBAdapter {
    provides read();
  always execution_time(read) < ExecutionTime;
  };
  Component UserManagement {
    provides authenticateUser();
  always execution_time(authenticateUser) <= ExecutionTime;
  };
  Component OggEncoder{
    provides encodeAudioData();
  always execution_time(encodeAudioData) <= ExecutionTime;
  };
  Component EncodingAdapter {
    provides processEncoding();
  always execution_time(processEncoding) <= ExecutionTime;
  };
  Component MySqlClient {
    provides storeAudioFile();
  always execution_time(storeAudioFile) <= ExecutionTime;
  };
  resource DB.canHandle (
    Set{Query(
      met = ResponseTime)});
  provides
    service implemented by MySqlClient {
      ExecutionTime < ResponseTime =>
      always response_time (storeAudioFile) < ResponseTime;
}
```

Listing 8.5: Web Audio Store Container Specifications via QML/CS

The container strategy defines how the query execution time for an operation of a component can be represented. This is achieved using a helper ExecutionTime. The ResponseTime is an upper bound for the component; the container declaration ensures

that query execution time for a component is not more than this value. The service part of the container expresses how the service represents the same query execution time operation by the component and this query execution time is constrained by the specification parameter ResponseTime.

8.2.4 System Designer's Perspective

The system designer's perspective covers the larger scope when specifications of different containers are combined together. It bears prime importance since different measurements need to be linked together if they require similar resources or if a component can provide more than one operations being exposed by a service. The system designer considers the integration perspectives of the application and uses QML/CS to specify system level non-functional behaviour of the application. The following system specification binds together the specifications from the declarations shown in previous sections.

Listing 8.6 shows system designer's perspective of the system specification. It indicates the relation between different instances of the system like a container, service offer on behalf of the container and the required resources. This is where the use of resources can be optimised because multiple components and services may require same resource (e.g. CPU) and their requests need to be added to scheduled tasks considering the resource capacity. It organises working combination of different elements in the form of defined container structures with specifying which component needs what resources. Lines 1 to 4 show the different elements that are part of the system and the resources available to the system. Lines 6 to 10 show the link of instances of container with specific components and the services being offered as part of the container.

```
1  System WebAudioStoreApplication {
2    instance ServiceMySqlClient MySqlClientService;
3    instance ComponentWebForm WebForm;
4    instance ComponentDBAdapter DBAdapter;
5    instance ComponentAudioStore AudioStore;
6    instance ComponentDBAdapter DBAdapter;
7    instance ComponentUserManagement UserManagement;
8    instance ComponentOggEncoder OggEncoder ;
9    instance ComponentEncodingAdapter EncodingAdapter;
10   instance ComponentMySqlClient MySqlClient;
11   instance ResourceMySqlDB DB ;
12   instance AudioSystemContainer (30) container;
13
14   container
15     uses WebForm, AudioStore, DBAdapter, UserManagement,OggEncoder,
     EncodingAdapter, MySqlClient, DB;
```

```
16   container
17     provides ServiceMySqlClient MySqlClientService;
18 }
```

Listing 8.6: Web Audio Store System Specifications via QML/CS

8.3 TLA+ Specification Generated

We present TLA+ translation for the selected specification elements. Theses specifications are being shown below in listings and remaining TLA+ specification can be found in Appendix A.2.

Listing 8.7 shows the TLA+ specification of *response time* module. It is equivalent to QML/CS specification of the *response time* discussed in listing 8.1. Lines 2 thru 9 extend *Real Time* module, line 7 makes use of the module of a *RT* context model. This concept of module is represented in QML/CS listing line 1 where the import statement 'in context RT' is used to link to the context model. Line 12 in TLA+ listing shows *OnInit* transition with some addition constraints, this is represented in QML/CS listing line 3. Line 17 represent the second transition named *RequestArrival* and its own constraints where this also has been specified in QML/CS listing in line 6. Line 21 also show the third transition named *StartRequest* and it is equivalent represented in line 9 of QML/CS listing. Line 24 shows the last transition named *FinishRequest* with its own constraints and this also shown in QML/CS listing line 11. The equivalent code of QML/CS listing also mentions the constraints on each transition for the variables *start* and *end*, which represent the start and end time of the operation and shows specifying the difference between end and start time of the last invocation of the single service operation. The *RTSpec* in this listing shows the state machine specification with a list of states that are part of state machine model of the measurement context.

```
1 ---------- MODULE response_time ----------
2 EXTENDS RealTime
3
4 VARIABLES inState, unhandledRequest
5 VARIABLES start, end
6
7 op == INSTANCE RT
8 ----------------------------
9 VARIABLE ResponseTime
10 VARIABLE hadOpCall
11 ----------------------------
12 OnInit == op!Init=>
13                   /\ response_time \in Real
14                   /\ ResponseTime= 0
```

```
15                    /\ hadOpCall= FALSE
16
17 OnRequestArrival == op!RequestArrival =>
18                                        /\ start = 0
19                                        /\ end = 0
20
21 OnStartRequest == op!StartRequest =>
22                                        /\ start'= now
23
24 OnFinishRequest == op!FinishRequest =>
25                                      /\ end'= now
26                                      /\ ResponseTime'= ResponseTime + end
27                                      /\ ResponseTime'= ResponseTime - start
28                                      /\ hadOpCall'= TRUE
29 RTSpec == /\ op!RT
30           /\ [][ OnInit/\OnRequestArrival /\ OnStartRequest /\
     OnFinishRequest]_<<ResponseTime, hadOpCall>>
31 ============================
```

Listing 8.7: The TLA+ specification for *response time*

Listing 8.8 shows the TLA+ specification of *execution time* module. It is equivalent to QML/CS specification of the *execution time* discussed in listing 8.2. Lines 2 thru 9 extend *Real Time* module, make use of the module of a *ET* context model. This concept of module is represented in QML/CS listing Line 1 where the import statement 'in context ET' is used to link to the context model. Line 12 in TLA+ listing shows *OnInit* transition with some addition constraints, this is represented in QML/CS listing line 3. Line 18 represent the second transition named *StartRequest* and its own constraints where this also has been specified in QML/CS listing in line 8. Line 22 also show the third transition named *FinishRequest* and it is equivalent represented in line 11 of QML/CS listing. Line 24 shows the fourth transition named *SwitchToOther* with its own constraints and this also shown in QML/CS listing line 14. Line 29 represents the transition named *SwitchBack* and it is equivalent to the QML/CS listing line 16. The corresponding code of QML/CS listing also shows the constraints on each transition for the variables *SegStart* and *AccExec*, which represents the start and accumulated execution time of the service execution. The *ETSpec* in this listing shows the state machine specification with a list of states that are part of state machine model of the measurement context.

```
1 ---------- MODULE execution time ----------
2 EXTENDS RealTime
3
4 VARIABLES inState, unhandledRequest
5 VARIABLES AccExec, SegStart
6
7 op1 == INSTANCE ET
8 ------------------------------
```

```
VARIABLE Execution_time
VARIABLE hadOpCall
----------------------------
OnInit == /\ execution_time \in Real
          /\ AccExec= 0
          /\ SegStart = 0
          /\ Execution_time =0
          /\ hadOp1Call= FALSE

OnStartRequest == op1!StartRequest =>
                            /\ SegStart'= now
                            /\ AccExec'= 0

OnFinishRequest == op1!FinishRequest =>
                            /\ Execution_time'= AccExec + now - SegStart
                            /\ hadOp1Call'= TRUE

OnSwitchToOther == op1!SwitchToOther =>
                            /\ AccExec'= AccExec + now - SegStart

OnSwitchBack == op1!SwitchBack =>
                            /\ SegStart'= now

ETSpec == /\op1!ET
          /\ [] [OnInit /\ OnStartRequest /\ OnFinishRequest /\
    OnSwitchToOther /\ OnSwitchBack]_<<Execution_time, hadOp1Call>>
============================
```

Listing 8.8: The TLA+ specification for *execution time*

8.4 Limitations of Evaluation

This section discusses the limitation of our evaluation from three perspectives. Subsection 8.4.1 discusses different approaches to empirical studies and assess our evaluation in this context. Subsection 8.4.2 discusses how we could have compared QML/CS to other languages. Subsection 8.4.3 discusses the limitation with regards to scope of our case study.

8.4.1 Empirical Studies

In any research, evaluation is of key importance. It not only concludes the objective of the research but also provides means for its validation. In this work, the evaluation has rather been of subjective nature because it only shows the usage of QML/CS

language and does not consider systematic experiments. This has been mainly due to time and resource constraints. However, we did explore the empirical strategies commonly used and applied to such research. In the following paragraphs, such empirical strategies and their scope is discussed. The four major empirical strategies are *experiment*, *case study*, *survey* and *post-mortem analysis* [106] as discussed below.

Experiment strategy expects the study to be conducted in a lab setting where statistics are noted and analysed [106]. The evaluation of our work does not come under the *experiment* strategy mainly because it was not conducted in any controlled environment like an experiment usually takes place. One way to conduct our work could have been to define different values for parameters like expressiveness, ease-of-use, time taken to write specification, complexity of the language and scope of the language to cover different specification concepts. Then an experiment could be conducted by selecting a set of developers and designers who specify one of their real time scenario they are implementing in an application. The observations from this set up could be analysed to reach a conclusion about the application of QML/CS in different experiment settings with each setting targeting a different set of NFPs.

The next approach is *case study* [106]. The *case study* is planned with steps and milestones and where data is collected in each step in order to reflect what was observed in previous step. Later analytical methods (e.g.regression) are applied to model the data. It is more flexible than conducting an experiment in terms of controlling the steps and applying more rigorous analytical models. Our evaluation considered a limited version of the *case study* approach without making any observations and conducting any analysis on the feedback. It also applied the language to one specific example whereas applying it to different real-life examples or domains could have provided more insight into the abilities of the language.

The third approach of *survey* generally needs a questionnaire to be designed and then handed over to relevant stakeholders to collect feedback. It also considers options like interviews, meetings, sessions and discussions before the questionnaire can be returned with feedback [106]. This approach however, demands a high volume of volunteers, contributors and hence, lacks in small scale research evaluations like ours. This approach is of quantitative nature whereas our evaluation is more of qualitative type. Moreover, it usually has a limited scope of applying something practically like *experiment* and *case study* approaches.

The *post-mortem* approach is another approach that is used either after the product is developed or retrospectively evaluation different parts of the product while it is being developed and partial milestones are achieved. It is more like a combination of *case study* and *survey* where analytical objectives are set first and later question-

naires or interviews are planned to collect feedback to achieve those objectives. It provides both qualitative and quantitative perspectives of the system for suggestions to improve it further [106]. This type of evaluation is usually conducted on a mature system where product development is well defined and is of commercial nature.

After exploring different options, we reached to this conclusion that probably the most suitable approach for our thesis evaluation should have been a strategy which is inclined towards *experiment* based approach but due to time and resources constraints we ended up in following a limited version of *case study* based strategy.

8.4.2 Comparing to other Languages

As we mentioned in Chapter 2, there are other generic languages like QML [40], CQML [3] or CQML+ [43]. The evaluation could have shown our language compared to these languages in two aspects;

1. QML/CS could have been compared with these languages in terms of its concepts and what concepts are available in these languages. It could have been compared with QML for its definition and values for a measurement as there is a difference in terms of determining its values and how they are derived. Another language that could be used for the concepts comparison is CQML+. It could be compared for *quality characteristics* concept with a measurement concept. However, the concepts in QML/CS are the same as in Zschaler TLA+ based-framework where such comparison has already been done by Zschaler [109]. Also, it would not produce any new information and would be merely repetition of the effort done already. This kind of evaluation may not be useful in terms of evaluating QML/CS as the foremost claim in this thesis is the level of ease and abstraction provided by QML/CS to write specifications.

2. QML/CS could have been compared with these languages in terms of usability and expressiveness. It could have been compared with CQML for syntax and semantic representation of concepts as there is major difference in the level of specification between QML/CS and CQML. QML/CS is a very high level language compared to natural language syntax of CQML. Another language that could be used for comparison is CQML+. It could be compared for ambiguity and its compatibility with the component-based systems. However, we did not have time and resources to conduct this kind of evaluation. That is why it was not addressed in depth part of this thesis. Therefore, this evaluation does not conduct any comparison with other generic languages.

8.4.3 Limited Scope of the Case Study

The evaluation focuses on only one case study where as it could have considered more than one case studies and in different domains. This could have given a better insight to the expressiveness and the applicabilities of QML/CS. It could also highlight some of the hidden issues in specification capabilities of QML/CS when focusing problems in different domains. In addition, the evaluation focuses only on individually measurable non-functional properties and highlights the limitation of QML/CS for specifying non-stochastic properties like usability. As a matter of fact, it evaluates only expressiveness of QML/CS and the ease with which it can be used for specification with application to one case study in a specific domain.

8.5 Summary

In this chapter, a detailed evaluation of the proposed QML/CS language has been provided. The evaluation provides a case study [11] to show the applicability of QML/CS in specifying a realistic application. A demonstration of how to apply QML/CS to specify a system is shown. A complete set of QML/CS models of the case study were provided along with discussing how these models were developed. The TLA+ specifications that are produced by the code generator were presented. It is shown, using a case study, that QML/CS language can be used to completely specify the non functional specification of a component based system. In the end, limitation of our evaluation is discussed where efforts can be made to explore the influence of the proposed language in much detail in future. The next chapter will conclude the study conducted during the course of this thesis, provide a summary and highlight the areas of future work.

9 Conclusion

This chapter will summarize the work done in this thesis for implementation of QML/CS language for NFPs of component-based systems. It mentions the problems faced in definition of such a specification language and how those problems were addressed. It lists the contributions made in this thesis with brief information about how does each contribution help solve the problems mentioned on Chapters 4 and 5. This chapter also discusses limitation of current work and future recommendations.

This thesis provides a usable generic high-level specifications language for NFPs of component-based systems. In component-based system, there is a number of players like designer, developer, quality engineer and deployment resource involved and a standard process is needed to communicate between them so that information about quality of the system can be measured and exchanged effectively. A formal and complete specification of NFPs is important not only for the the development and testing of a software system but also for its operation in the deployment environment. Therefore a usable, precise and formal specification of NFPs of component-based system is important in component-markets.

Nevertheless, existing approaches of specifying NFPs lack usability and formality while others are formal but the low level formalisation limit their usability. In addition, are some approaches which are formal and usable but they're not suitable for component-based system concepts. Moreover, there are other usable and formal approaches, however, they cannot be used to model NFPs generically. In this thesis, we first identified a gap, which is a need of generic usable language to specify NFPs of component-based system. We found that there is a potential language named QML/CS, however, as it exists it is not formally defined. Therefore, we take inspiration of the work done in [110], and provided a formal definition of QML/CS language through creation of a meta-model. This involves employing techniques from model-driven engineering such as deep meta-modelling, domain-specific language workbenches, code generation and weaving models to answer the research questions mentioned in Chapter 1. A semantic definition for QML/CS is provided by translating its specification

into TLA+ using a model transformation approach. We have also demonstrated its usability by modelling an industrial application using QML/CS and show the benefit of having a language like QML/CS by comparing it with TLA+ as discussed in Chapter 8. Prototype implementation has been developed for QML/CS proposed in this thesis. The purpose of this prototype is to support the usability of QML/CS and setting a base to integrate QML/CS in standardised tools.

9.1 Addressing the Research Questions

1. *Is it possible to specify the QML/CS specification language using a meta-model, if so how?*

   The first research question is addressed by showing the ways of defining QML/CS presented in this thesis. An initial attempt was carried out to model QML/CS using UML, however, the conventional UML modelling language does not support more than two levels of modelling and it is limited to the concept of a class and its instance. Thus, an entity that exists as a class and instance at the same time can not be represented. This is discussed in more details in Chapter 4. In order to successfully model such a requirement where the entities can be specified at multiple levels and their existence depends on the relationship they have with their entities, we required a modelling technique that can represent more than one existence of same entity based on the role of that entity in that specific context. We discovered that an existing modelling technique called Clabject comes handy in modelling such entities and gives discrete representation to their both roles of a class and an instance. Therefore, a second attempt made to model QML/CS by applying deep meta-modelling that supports Clabject technique. The result of applying this technique is discussed in Chapter 4. Although in [21], it is suggested that the Meta Object Facility (MOF) can either be used to extend the UML meta-model or define the meta-model of the new modelling language. It is however, noted from this research that such a benefit is not likely for some modelling languages like QML/CS with the obvious limitation of defining more than two levels as discussed in Chapter 4.

2. *Whether a usable QML/CS specification language can be defined?*

   To answer this research question, a realistic application of an Audio Store was modelled using QML/CS and the measurements for both component and service operations were specified. The experiment shows the feasibility of QML/CS to describe realistic applications and also the support to which the desired NFPs can be defined as required.

9.2 Contribution to the Body of Knowledge

The aim of this thesis was to present a quality modelling language for specifying NFPs of component-based system. The following contributions have been made by this thesis points:

- Major Contributions:
  - *The thesis provides a novel specification language for the formal specification of NFPs of component-based systems (Language Definition for QML/CS).*
    The provision of a formal specification language for NFPs is one of the key primary contributions made by this thesis. It includes definition of language grammar, integrating parsers to convert this raw grammar into a usable language with strong semantics, make it compatible to be used on standard language tools, develop a prototype tool to demonstrate the usage and building a transformation to TLA+. Chapter 3 presents the main concepts of QML/CS and includes the meta-classes and examples of each concept.
  - *Applying deep meta-Modelling to define QML/CS.*
    The dual-instantiation deficiency of conventional modelling languages like UML has been solved by applying deep meta-modelling. Chapter 4 highlights the deficiency of modelling languages like UML in modelling QML/CS. It also provides details on how the deep meta-modelling can be applied to achieve dual instantiation of the entities to implement multi-level modelling, which is a key feature of QML/CS. This is a general issue with generic languages like QML/CS. As such, the language provides a potentially interesting case study for deep-meta-modelling research.

  - *The ability to capture and validate simulations between state machines in a mapping model.*
    A mapping model has been specified in this thesis to enable mapping between context model and the relevant application model. It allows specification of measurements independent of concrete application. Chapter 5 shows how a measurement can be applied to concrete application, which is related to establishing a mapping between context and application models of QML/CS. It also presents a solution of weaving model that addresses this problem. Thesis shows the mapping strategy implementation between the model of QML/CS and any given operation in an actual application through the state machine models and class diagrams, that is part of model mapping.

- Secondary Contributions:
  - *The thesis defines the Semantics Translation to TLA+.*
    Transformation is established from a high level specification of QML/CS into TLA+ specification. It helps to map through the QML/CS specification and provided context or application model to produce an equivalent of TLA+ specifications as shown in Chapter 7. The benefit of this contribution is that QML/CS language can be compared with other languages and frameworks and usage of QML/CS is clearly justified when it comes to ease of use, complexity and the time to complete the specification.
  - *A working prototype as a basis for the integration of QML/CS in future language specification tools.*
    A working prototype is implemented to support QML/CS integration as shown in Chapter 6. It also provides a basis for future work where smart tools can be implemented to allow users to specify NFPs using QML/CS.
  - *Integrating OCL into QML/CS Grammar.*
    QML/CS allows the use of OCL to specify constraints in the form of expressions and integration of QML/CS with OCL is implemented as part of the language. It adds benefits of OCL to QML/CS language and also makes it easy to extend OCL to add further features to QML/CS in future. The complete strategy of integrating OCL into QML/CS is presented in Chapter 6.

9.3 Future Recommendations and Lessons Learned

This section discusses potential directions for future work and recommendations. It details some possible future directions of research based on the work presented in this thesis. The possibilities are split into the following categories:

- *Implementation*
  Improving the implementation of the language and extending it to cover other NFPs. This thesis has focused on measurable NFPs of component-based systems and specification of other NFPs needs research on how they can be specified. The current language implementation has provision of linking with context models and therefore new context models can be designed to add support for other NFPs that are not discussed in current work.

- *Inter-Component Integration*
  The non-function aspect of component to component integration and coordination is another domain where research would be required to extend the existing

context models. This will help represent the inter-component integration and coordination as context models and then QML/CS can help specifying those interfaces between the components.

- *Support for Non-Component-Based Systems*
  The current implementation specifically covers component-based systems and has not discussed about systems that are designed with non component-based approach. Although specification of measurements can be used for non component-based systems as well but the concepts of components and services limit their use. More abstract concepts can be added so that non component-based systems can also be modelled for their NFPs using QML/CS.

- *Tool Support for QML/CS*
  This is an important area where work is needed so that use of QML/CS can be extended to the users who are already using tools like Palladio for specification of NFPs. The implementation of QML/CS plugins for these tools will give users a choice to use QML/CS for specification.

- *Extended Application*
  Although current implementation of QML/CS has been applied to realistic application for evaluating usability of QML/CS, a diverse type of applications should be specified using QML/CS so that gaps in its implementation can be discovered based on different usage. It will help improve the language to handle not only different NFPs but also cover different types of projects even in the same domain of component-based systems.

The future research in this area would be helped with experiences in this thesis that posed challenges to achieve the implementation of QML/CS. Designing a specification language like QML/CS needs to be done in a complete cycle that has stages to complete before work can be started on the next stage. Writing DLS of a specification language needs a complete knowledge of what is to be provided as features of the language so that syntax and semantic controls can be specified in DSL. It also needs the rules for validating that DSL to be specified so that any generic tool like xtext can derive the evaluation criteria from that DSL and then help build a language on top of it. Completion of DSL for a language is itself an iterative process because many things are highlighted when applying the DSL and then another process to improve DSL starts to fill the missing links. Once this is done then need arises for providing an IDE where this grammar can be exploited to convert it into a specification language. The NFPs that can be supported by the language depends not only on specification but its semantics are controlled by DSL as well. Even if the language is able to specify NFPs; one important step is validation of language if it is compatible to standards like TLA+ to ensure that the language represents the concepts effectively. It is such

a linked and iterative exercise that one mistake at any level can propagate to create a language that may not be practically usable.

The limitation of existing MOF meta-model for its support restricted to two levels of modelling is a potential challenge for building domain specific languages because they might need more detailed expression of the relationships needing more than two levels of modelling. Although there was an option to extend UML modelling capacity but that may have the two-level modelling restriction at its core and therefore will not serve the purpose. That is why we decided to use multi-level modelling so that the level and scope of expression of the language is flexible. This thesis has shown the possible problems with two-level modelling and they have been discussed in detail. The thesis has also presented the benefits of multi-level modelling to address the challenges but more research is needed to confirm the compatibility of multi-level modelling to design domain specific languages where it is applied to more generic concepts and real scenarios. This further research will evaluate application of multi-level modelling to different domains and concepts within the same domain for which the language is designed.

It is an important consideration that domain specific languages confirm to the standards like OCL so that they take advantage of the existing concepts. The implementation of QML/CS needed to integrate with OCL to use the features like OCL expressions to define constraints and requirements for different features in the language like resources and their task allocation. Building a bridge component between QML/CS and OCL was not part of the thesis and it was concluded that it will also take the focus away from the original purpose of the thesis; building the QML/CS language itself. Also the integration of QML/CS editor with OCL would need further integration components just for the sake of using OCL in the language. After careful consideration of these extra requirements and their impact on the timeline of developing the QML/CS language itself, it was decided to embed the OCL grammar within the grammar of QML/CS so that the extension of concepts like FeatureCallExpression can be implemented without changing the basic structure of QML/CS and without needing any external components to facilitate the language or its editor. This has potential issues of updating the OCL if a new version is available with better concepts as that grammar will have to be embedded into QML/CS for next releases to make it compatible with latest changes in OCL.

It was important to design QML/CS language in a manner so that it can be applied for small, medium and large level component-based systems. This is because if the language is applicable to only small scale projects or case studies then its scope of being useable is very limited. Compared to languages like TLA+, which generate a substantial amount of code and specification for medium or large scale projects,

QML/CS uses higher level of abstraction to enable specification of larger systems with less code and better understanding and linking of concepts required for the system in consideration. This was one of the key observations and that should be considered when designing and specification language because larger the code is, more complex it is to establish link between different concepts of the system and we can easily lose track of where it is going. Although there is more research needed to find a balance between higher level of abstraction and giving ability to the user to control the level of abstraction but QML/CS implements a first level of balance between the two. Further evaluation and application to systems of different size can provide input on whether it is enough or it needs more work to make the abstraction more useful.

Bibliography

[1] *Software Engineering - Product Quality, ISO/IEC 9126-1*, 2001.

[2] Object Management Group, Unified Modeling LanguageTM (OMG UML), Superstructure Version2.2. Technical Report formal/2009-02-02, 2009.

[3] J. Ø. Aagedal. *Quality of Service Support in Development of Distributed Systems*. PhD thesis, University of Oslo, 2001.

[4] M. Abadi and L. Lamport. An old-fashioned recipe for real time. *ACM ToPLaS*, 16(5):1543–1571, Sept. 1994.

[5] D. H. Akehurst, S. Zschaler, and W. G. J. Howells. OCL: Modularising the Language. *ECEASST*, 9, 2008.

[6] AMW (ATLAS Model Weaver website). http://eclipse.org/gmt/amw/, 2007.

[7] T. Asikainen and T. Männistö. Nivel: a metamodelling language with a formal semantics. *Software and System Modeling*, 8(4):521–549, 2009.

[8] C. Atkinson and T. Kühne. Rearchitecting the UML Infrastructure. *ACM Trans. Model. Comput. Simul.*, 12(4):290–321, Oct. 2002.

[9] P. Banerjee and A. Sarkar. Formal specification of extra-functional properties for component based system. In *Proceedings of the 2014 International Conference on Information and Communication Technology for Competitive Strategies*, ICTCS '14, pages 28:1–28:7, New York, NY, USA, 2014. ACM.

[10] P. Banerjee and A. Sarkar. Z specification of component based software, 2014.

[11] S. Becker, H. Koziolek, and R. Reussner. The palladio component model for model-driven performance prediction. *J. Syst. Softw.*, 82(1):3–22, Jan. 2009.

[12] J. Beézivin, G. Hillairet, F. Jouault, I. Kurtev, and W. Piers. Bridging the MS/DSL Tools and the Eclipse Modeling Framework. In *In: Proceedings of the International Workshop on Software Factories at OOPSLA 2005*, 2005.

[13] A. Bertolino and R. Mirandola. CB-SPE Tool: Putting component-based performance engineering into practice. In *7th International Symposium on Component-Based Software Engineering (CBSE 2004*, pages 233–248. Springer, 2004.

[14] L. Bettini. *Implementing Domain-Specific Languages with Xtext and Xtend*. 2013.

[15] A. Beugnard, J.-M. Jézéquel, N. Plouzeau, and D. Watkins. Making components contract aware. *Computer*, 32(7):38–45, July 1999.

[16] J. Bézivin. Model engineering for software modernization. In *WCRE*, page 4 only, 2004.

[17] J. Bézivin, V. Devedzic, D. Djuric, J.-M. Favreau, D. Gasevic, and F. Jouault. An m3-neutral infrastructure for bridging model engineering and ontology engineering. In Springer, editor, *Int. Conf. on Interoperability of Enterprise Software and Applications (INTEROP-ESA)*, pages 159–171, Geneva, Switzerland, 2005. Springer. 1-84628-151-2.

[18] J. Bézivin and O. Gerbé. Towards a Precise Definition of the OMG/MDA Framework. In *ASE*, pages 273–280, 2001.

[19] B. W. Boehm, J. R. Brown, and H. Kaspar. *Characteristics of Software Quality. Vol. 1.* TRW series of software technology. North-Holland Publishing, 1978.

[20] E. Bondarev, M. R. V. Chaudron, and P. H. N. de With. Compositional performance analysis of component-based systems on heterogeneous multiprocessor platforms. In *EUROMICRO-SEAA*, pages 81–91. IEEE, 2006.

[21] A. D. Brucker and J. Doser. Metamodel-based uml notations for domain-specific languages, 2007.

[22] T. Bures, J. Carlson, I. Crnkovic, S. Sentilles, and A. Vulgarakis. Procom - the progress component model reference manual, version 1.0. Technical Report ISSN 1404-3041 ISRN MDH-MRTC-230/2008-1-SE, June 2008.

[23] X. Cai, M. R. Lyu, K.-F. Wong, and R. Ko. Component-based software engineering: Technologies, development frameworks, and quality assurance schemes. In *Proceedings of the Seventh Asia-Pacific Software Engineering Conference*, APSEC '00, Washington, DC, USA, 2000. IEEE Computer Society.

[24] M. V. Cengarle and A. Knapp. Towards OCL/RT. In *FME*, volume 2391 of *Lecture Notes in Computer Science*, pages 390–409. Springer, 2002.

[25] J. Cheesman and J. Daniels. *UML Components: a simple process for specifying component-based software.* Addison-Wesley, 2000.

[26] L. Chung and J. C. Prado Leite. Conceptual modeling: Foundations and applications. chapter On Non-Functional Requirements in Software Engineering, pages 363–379. Springer-Verlag, Berlin, Heidelberg, 2009.

[27] T. Cleenewerck and I. Kurtev. Separation of concerns in translational semantics for dsls in model engineering. In *Proceedings of the 2007 ACM Symposium on Applied Computing*, SAC '07, pages 985–992. ACM, 2007.

[28] C. Consel and O. Danvy. Static and dynamic semantics processing. In *Proceedings of the 18th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '91, pages 14–24. ACM, 1991.

[29] M. Dahchour. Formalizing materialization using a metaclass approach. In *Advanced Information Systems Engineering, 10th International Conference CAiSE'98, Pisa, Italy, June 8-12, 1998, Proceedings*, pages 401–421, 1998.

[30] J. de Lara and E. Guerra. Deep meta-modelling with metadepth. In *Proceedings of the 48th International Conference on Objects, Models, Components, Patterns*, TOOLS'10, pages 1–20, Berlin, Heidelberg, 2010. Springer-Verlag.

[31] J. de Lara, E. Guerra, and J. S. Cuadrado. When and how to use multilevel modelling. *ACM Trans. Softw. Eng. Methodol.*, 24(2):12, 2014.

[32] E. Demairy, E. Anceaume, and V. Issarny. On the correctness of multimedia applications. In *11th Euromicro Conference on Real-Time Systems (ECRTS 1999), 9-11 June 1999, York, England, UK, Proceedings*, pages 226–233, 1999.

[33] Dimitrios S. Kolovos, Richard F. Paige, Louis M. Rose, and Fiona A. C. Polack. *Epsilon*. 2008.

[34] F. Duran, A. Moreno-Delgado, F. Orejas, and S. Zschaler. Amalgamation of domain specific languages. *Journal of Logical and Algebraic Methods in Programming*, 2015.

[35] F. Durán, S. Zschaler, and J. Troya. On the reusable specification of non-functional properties in DSLs. In K. Czarnecki and G. Hedin, editors, *Proc. 5th Int'l Conf. on Software Language Engineering (SLE'12)*, volume 7745 of *LNCS*, pages 332–351. Springer, 2013.

[36] Eclipse Foundation. Eclipse and Eclipse Foundation, Online; accessed 20-Jan-2014.

[37] S. Efftinge and M. Völter. oAW xText: A framework for textual DSLs. In *Eclipse con Summit Europe 2006*, 2006.

[38] M. Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3 edition, 2003.

[39] X. Franch. Systematic formulation of non-functional characteristics of software. In *Proc. 3rd Int'l Conf. on Requirements Engineering*, pages 174–181. IEEE Computer Society, 1998.

[40] S. Frolund and J. Koistinen. QML: A language for quality of service specification. Technical Report HPL-98-10, Hewlett-Packard Software Technology Laboratory, February 1998.

[41] O. M. Group. Common Object Request Broker Architecture (CORBA) Specification, Version 3.3. OMG Document, November 2012.

[42] X. Gu, K. Nahrstedt, W. Yuan, D. Wichadakul, and D. Xu. An xml-based quality of service enabling language for the web. *J. Vis. Lang. Comput.*, 13(1):61–95, 2002.

[43] H. Härtig, S. Zschaler, M. Pohlack, R. Aigner, S. Göbel, C. Pohl, and S. Röttger. Enforceable component-based realtime contracts – supporting realtime properties from software development to execution. *Springer Real-Time Systems Journal*, 35(1), Jan. 2007.

[44] F. Heidenreich, J. Johannes, S. Karol, M. Seifert, and C. Wende. Model-based language engineering with emftext. In R. LÃd'mmel, J. Saraiva, and J. Visser, editors, *GTTSE*, volume 7680 of *Lecture Notes in Computer Science*, pages 322–345. Springer, 2011.

[45] G. T. Heineman and W. T. Councill, editors. *Component-based Software Engineering: Putting the Pieces Together*. Addison-Wesley Longman Publishing Co., Inc., 2001.

[46] K. T. G.-P. C. Henderson-Sellers B, Atkinson C. Understanding metamodelling. *= Tutorial in 22nd International Conference on Conceptual Modeling ER2003, year = 2003, url = Retrieved November 20, 2004 from http://www.er.byu.edu/er2003/slides/ R2003T1HendersonSellers.pdf.*

[47] international. International standard ISO/IEC 9126. information technology – software product evaluation – quality characteristics and guidelines for their use.

[48] Information technology – quality of service: Framework. ISO/IEC 13236:1998, ITU-T X.641, 1998.

[49] J. Miller and J. Mukerji. Mda guide version 1.0.1, 2003.

[50] M. Jackson. Some basic tenets of description. *Software and System Modeling*, 1(1):5–9, 2002.

[51] M. Jarke, R. Gallersdörfer, M. A. Jeusfeld, and M. Staudt. Conceptbase - A deductive object base for meta data management. *J. Intell. Inf. Syst.*, 4(2):167–192, 1995.

[52] K. Jeźek. *Extra-Functional Properties Support For a Variety of Component Models*. PhD thesis, University of West Bohemia, 2012.

[53] K. Jeźek and P. Brada. Formalisation of a generic extra-functional properties framework. In L. Maciaszek and K. Zhang, editors, *Evaluation of Novel Approaches to Software Engineering*, volume 275 of *Communications in Computer and Information Science*, pages 203–217. Springer Berlin Heidelberg, 2013.

[54] X. Jia. ZTC: A type Checker for Z Notation User Guide. Technical report, August 1998.

[55] G. Karsai, A. Misra, J. Sztipanovits, Á. Lédeczi, and M. Moore. Model-integrated system development: models, architecture, and process. In *COMPSAC*, pages 176–181, 1997.

[56] A. Kleppe, J. Warmer, and W. Bast. *MDA Explained: The Model Driven Architecture(TM): Practice and Promise*. Addison-Wesley Professional, 2005.

[57] A. G. Kleppe. A language description is more than a metamodel. In *Fourth International Workshop on Software Language Engineering, Nashville, USA*. megaplanet.org, October 2007.

[58] S. Kounev, F. Brosig, and N. Huber. The Descartes Modeling Language. Technical report, Department of Computer Science, University of Wuerzburg, October 2014.

[59] F. Krikava and P. Collet. On the use of an internal DSL for enriching EMF models. In *Proceedings of the 12th Workshop on OCL and Textual Modelling, Innsbruck, Austria, September 30, 2012*, pages 25–30, 2012.

[60] T. Kuehne and D. Schreiber. Can programming be liberated from the two-level style: Multi-level programming with deepjava. *SIGPLAN Not.*, 42(10):229–244, Oct. 2007.

[61] D. D. Lamanna, J. Skene, and W. Emmerich. Slang: A language for defining service level agreements. In *Proceedings of the The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems*, FTDCS '03, pages 100–, Washington, DC, USA, 2003. IEEE Computer Society.

[62] L. Lamport. *Specifying Systems, The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2002.

[63] Lawrence Chung and Julio Cesar Sampaio do Prado Leite. On Non-Functional Requirements in Software Engineering. In *Conceptual Modeling: Foundations and Applications - Essays in Honor of John Mylopoulos*, pages 363–379, 2009.

[64] C. Lee. *On Quality of Service Management.* PhD thesis, Carnegie Mellon University, Aug. 1999.

[65] Louis M. Rose and Richard F. Paige and Dimitrios S. Kolovos and Fiona Polack. The Epsilon Generation Language. In *Model Driven Architecture - Foundations and Applications, 4th European Conference, ECMDA-FA 2008, Berlin, Germany, June 9-13, 2008. Proceedings*, pages 1–16, 2008.

[66] M. W. Maier, D. Emery, and R. Hilliard. Ansi/ieee 1471 and systems engineering. *Syst. Eng.*, 7(3):257–270, Sept. 2004.

[67] R. Malan and D. Bredemeyer. Defining non-functional requirements. Bredemeyer Consulting, White Paper. http://www.bredemeyer.com/papers.htm, 2001.

[68] Meinte Boersma. Multi-level modeling: what, why and how, October 4, 2014.

[69] B. Meyer. *Object-oriented Software Construction (2nd Ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1997.

[70] Microsoft. COM+ (Component Services), version 1.5, Final release, 2003.

[71] S. Microsystems. Enterprise javabeans specification, version 3.2, final release, May 2013.

[72] M. Mohammad and V. Alagar. Tadl - an architecture description language for trustworthy component-based systems. In *Proceedings of the 2nd European Conference on Software Architecture*, ECSA '08, pages 290–297. Springer-Verlag, 2008.

[73] P.-A. Muller, F. Fondement, B. Baudry, and B. Combemale. Modeling Modeling Modeling. *Journal of Software and Systems Modeling (SoSyM)*, 11(3):347–359, 2012.

[74] J. Mylopoulos. Metamodeling. University Lecture, 2004.

[75] J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis. Telos: Representing knowledge about information systems. *ACM Trans. Inf. Syst.*, 8(4):325–362, 1990.

[76] Object and Reference Model Subcommittee of the Architecture Board. A Proposal for an MDA Foundation Model, ormsc/05-04-01. Technical report, Object Management Group, Apr. 2005.

[77] Object Management Group. Model Driven Architecture - A Technical Perspective.
urlhttp://www.omg.org/docs/ormsc/01-07-01.pdf, July 2001.

[78] Object Management Group. UML profile for schedulability, performance, and time specification. OMG Document, Mar. 2002.

[79] Object Management Group. UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE). OMG Document, Feb. 2005.

[80] G. Om. XML metadata interchange (XMI) specification. Technical report, Object Management Group, 2002.

[81] OMG. *Model Driven Architecture (MDA) Guide*, 2003. OMG doc. ab/2003-06-01.

[82] OMG. Unified Modeling Language: Superstructure, version 2.1.1. 2007.

[83] OMG. OMG Object Constraint Language (OCL), Version 2.3.1. Technical report, Object Management Group, January 2012.

[84] OMG. Pending Issues sent to the OMG Finalization Task Force corresponding to the Schedulability Performance and Time, profile, accessed June 2014.

[85] G. D. Plotkin. The origins of structural operational semantics. *Journal of Logic and Algebraic Programming*, 60:60–61, 1981.

[86] R. H. Reussner, H. W. Schmidt, and I. H. Poernomo. Reliability prediction for component-based software architectures. *J. Syst. Softw.*, 66(3):241–252, June 2003.

[87] S. Röttger and S. Zschaler. CQML$^+$: Enhancements to CQML. In J.-M. Bruel, editor, *Proc. 1st Int'l Workshop on Quality of Service in Component-Based Software Engineering, Toulouse, France*, pages 43–56. Cépaduès-Éditions, June 2003.

[88] S. Röttger and S. Zschaler. Model-driven development for non-functional properties: Refinement through model transformation. In *Proceedings of the «UML» Conference, Lisbon, Portugal*, volume 3273 of *Lecture Notes in Computer Science*. Springer, 2004.

[89] S. Röttger and S. Zschaler. A software development process supporting non-functional properties. In *Proc. IASTED Int'l Conf. on Software Engineering (IASTED SE 2004)*. ACTA Press, 2004.

[90] S. Röttger and S. Zschaler. Tool support for refinement of non-functional specifications. *Software and System Modeling*, 6(2):185–204, 2007.

[91] D. A. Schmidt. Programming language semantics. In *Encyclopedia of Computer Science*, pages 1463–1466. John Wiley and Sons Ltd.

[92] D. A. Schmidt. *Denotational Semantics: A Methodology for Language Development*. William C. Brown Publishers, 1986.

[93] E. Seidewitz. What models mean. *IEEE Softw.*, 20(5):26–32, Sept. 2003.

[94] S. Sentilles, P. Stepan, J. Carlson, and I. Crnkovic. Integration of extra-functional properties in component models. In G. A. Lewis, I. Poernomo, and C. Hofmeister, editors, *CBSE*, volume 5582 of *Lecture Notes in Computer Science*, pages 173–190. Springer, 2009.

[95] J. Skene, D. D. Lamanna, and W. Emmerich. Precise service level agreements. In *Proc. 26th Int'l Conf. on Software Engineering (ICSE'04)*, pages 179–188. IEEE Computer Society, May 2004.

[96] C. U. Smith. *Performance Engineering of Software Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1990.

[97] I. Sommerville. *Software engineering (5. ed.)*. Addison-Wesley, 1996.

[98] J. M. Spivey. *The Z Notation: A Reference Manual*, second edition edition, 2001.

[99] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. *EMF: Eclipse Modeling Framework 2.0*. Addison-Wesley Professional, 2nd edition, 2009.

[100] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2002.

[101] The Eclipse Foundation. Essentialocl.xtext, Online; last accessed 20-03-2015.

[102] The World Wide Web Consortium (W3C). Web Services Description Language (WSDL)2.0, 2007.

[103] J. Troya, J. E. Rivera, and A. Vallecillo. Simulating domain specific visual models by observation. In *Proc. 2010 Spring Simulation Multiconference*, SpringSim '10, pages 128:1–128:8, New York, NY, USA, 2010. ACM.

[104] D. Varró and A. Pataricza. VPM: A visual, precise and multilevel metamodeling framework for describing mathematical domains and UML (the mathematics of metamodeling is metamodeling mathematics). *Software and System Modeling*, 2(3):187–210, 2003.

[105] Volz, Bernhard and Jablonski, Stefan. Towards an open meta modeling environment. In *Proceedings of the 10th Workshop on Domain-Specific Modeling*, DSM '10, pages 17:1–17:6, New York, NY, USA, 2010. ACM.

[106] C. Wohlin, M. Höst, and K. Henningsson. *Empirical Methods and Studies in Software Engineering: Experiences from ESERNET*, chapter Empirical Research Methods in Software Engineering, pages 7–23. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[107] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[108] P. Ziemann and M. Gogolla. Ocl extended with temporal logic. In M. Broy and A. V. Zamulin, editors, *Ershov Memorial Conference*, volume 2890 of *Lecture Notes in Computer Science*, pages 351–357. Springer, 2003.

[109] S. Zschaler. *A Semantic Framework for Non-functional Specifications of Component-Based Systems*. Dissertation, Technische Universität Dresden, Dresden, Germany, Apr. 2007.

[110] S. Zschaler. Formal specification of non-functional properties of component-based software systems. *Software and System Modeling*, 9(2):161–201, 2010.

[111] S. Zschaler, D. S. Kolovos, N. Drivalos, R. F. Paige, and A. Rashid. Domain-specific metamodelling languages for software language engineering. In *Proceedings of the Second International Conference on Software Language Engineering*, SLE'09, pages 334–353, Berlin, Heidelberg, 2010. Springer-Verlag.

A Appendix

A.1 Specifications for Case Study

A.1.1 Context Models for Delta Time and Data Rate

The specification of two measurements *DeltaTime* and *DataRate* are shown in listings A.1 and A.2. It indicates that both *DeltaTime* and *DataRate* are importing their context and showing the time events that should be considered in their calculation. The *DeltaTime* defines the time between calls made to an operation in the system and gives information about how frequent an operation call is made. The *DataRate*, as its name indicates, measures the rate at which date emission takes place between two operation calls and shows how active the component is when the call is made from one call to the next one.

The listing A.1 starts (Lines 1 thru 9) with the definition of measurement *DeltaTime*, which is defined for single service operation *op* as time between two requests to know the frequency of this measurement being called by the environment. The listing shows the two transitions imported from the context model *InRT* so that *DeltaTime* specific variables *LastDeltaTime* and *StartDelta* can be defined and assigned values to calculate the output of the measurement.

```
1 in context InRT;
2 declare measurement Real DeltaTime (ServiceOperation op){
3 On op.Init update
4             LastDeltaTime=0;
5             StartDelta = now;
6 On op.RequestArrival update
7                       LastDeltaTime= now -StartDelta;
8                       StartDelta = now;
```

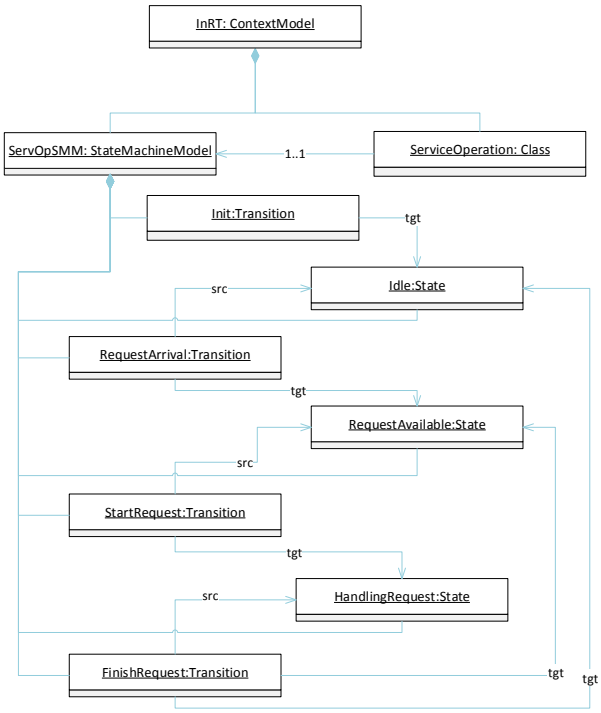

Figure A.1: Context Model for Delta Time (InRT).

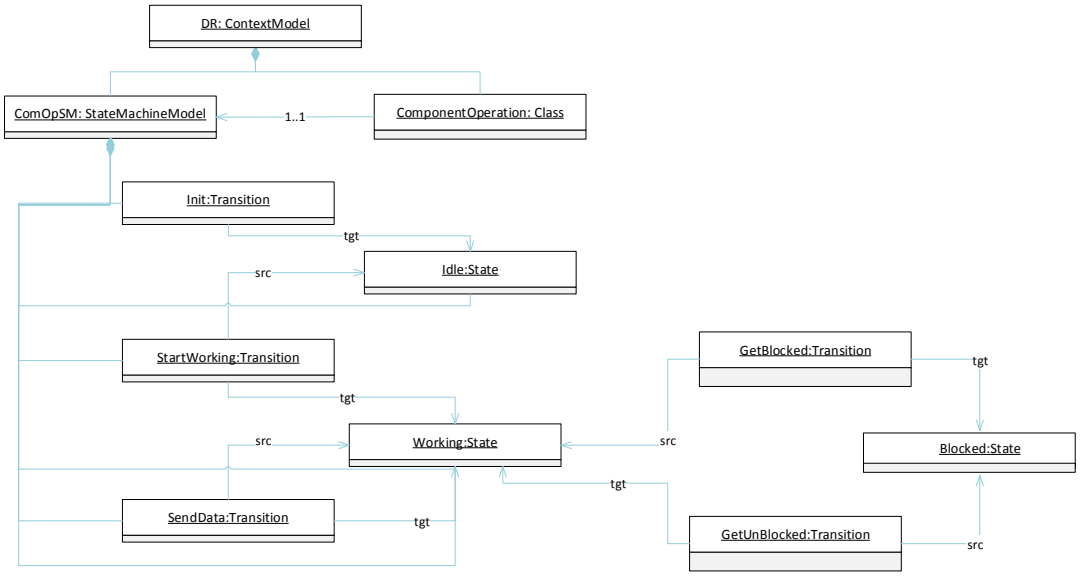

Figure A.2: Context Model for Date Rate (DR).

```
}
```

Listing A.1: The QML/CS specification for *Delta Time*

The listing A.2 from (Lines 1 thru 15) with the definition of measurement *DataRate* indicates the data generation of a component by measuring the amount of data emissions between two successive measurement calls. It uses transitions from the context model *DR* to specify different events like *Start*, *AccInterval* and *LastInterval* attached with those transitions. These events end up in changing values for the variables defined for measurement and then used for calculation of the measurement output.

```
in context DR;
declare measurement Real DataRate (ComponentOperation op){
On op.Init update
            AccInterval = 0;
            LastInterval = 0;
On op.StartWorking update
                    Start = now;
                    AccInterval = 0;
On op.GetBlocked update
                  AccInterval= AccInterval + now - Start;
On op.GetUnBlocked update
                    Start= now;
On op.DoSendData update
                  LastInterval= LastInterval + now - Start;
                  Start= now;
                  AccInterval=0;
}
```

Listing A.2: The QML/CS specification for *Data Rate*

A.1.2 Application models

we present the remaining application models of the case study as shown in Figs A.3, A.4, A.5, A.6, A.7 and A.8.

A.1.3 Application Specification

Further to our discussion in 8.2.2, line 9 in the listing 8.3 shows the validation of the *uploadFile()* operation is achieved by loading the *WebFormModel* and *ET* context model so that the mapping between those models can be established. This mapping is established through the mapping model named as *UFOp2EtMapping* as shown in Fig A.9. The *COpClassMapping* controls the mapping between source and target

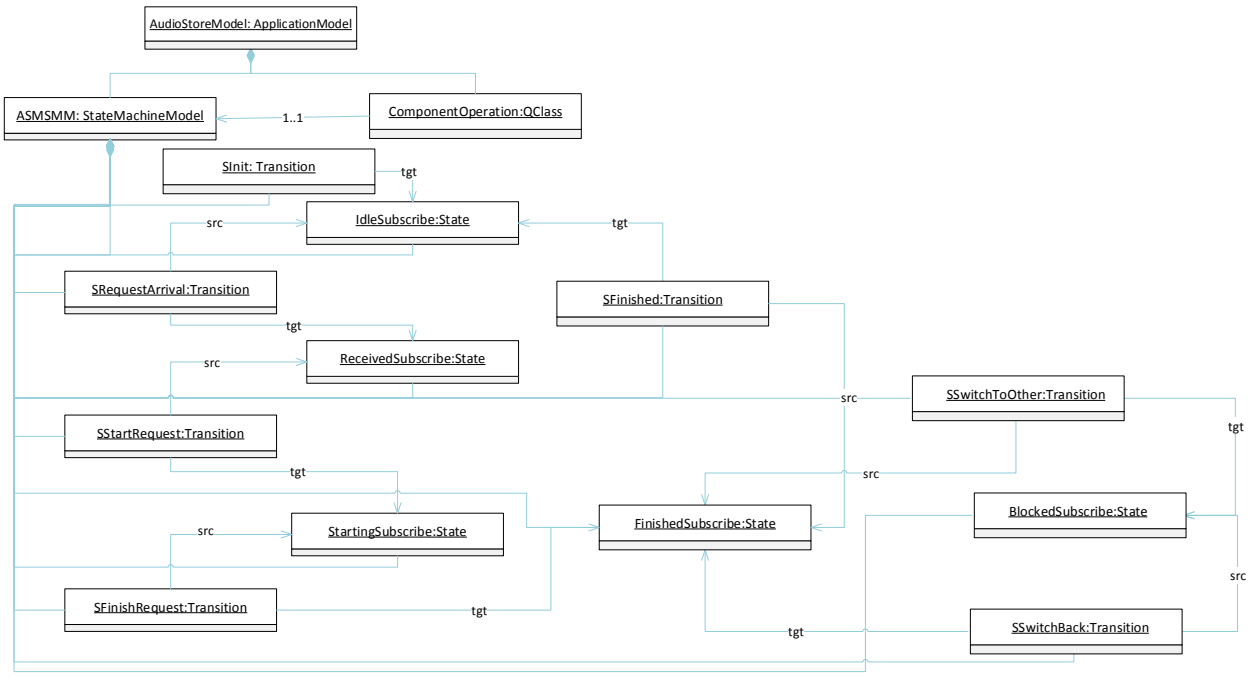

Figure A.3: Application Model for Audio Store Component.

Component Operation classes in application model and context model respectively. *StateMachineModelMapping* controls the mapping between source and target state machine model named *CSSMMStateMachineModelMapping* that contains a list of all the states and transitions that belong to the classes in the *WebFormModel* and *ET* context model. The *StateMapping* and *TransitionMapping* control mapping of each state and transition in state machine model of the *WebFormModel* to a state and transition in state machine model of the *ET* context model as required by the *StateMachineModelMapping*.

As discussed in Chapter 5, Fig A.9 also shows set of states and transitions; and the mapping instances that connect the states and transitions between application and context models. The equation one requires that for each states in state machine model of *WebForm* application model, there is at least one state in the *ET* context model to which it can be mapped. We can see in Fig A.9 that each of the states like *UFIdle*, *UFRequestAvailable*, *UFHandlingRequest*, *UFSwitchToOther* and *UFSwitchBack* are linked to relevant states in the *ET* context model via state mapping objects like *IdleSM*, *RequestAvailSM* and *HandlingRequestSM*. The second equation requires that for each transition in state machine model of *WebForm* application model, there is at least one transition in the *ET* context model to which it can be mapped. It is also

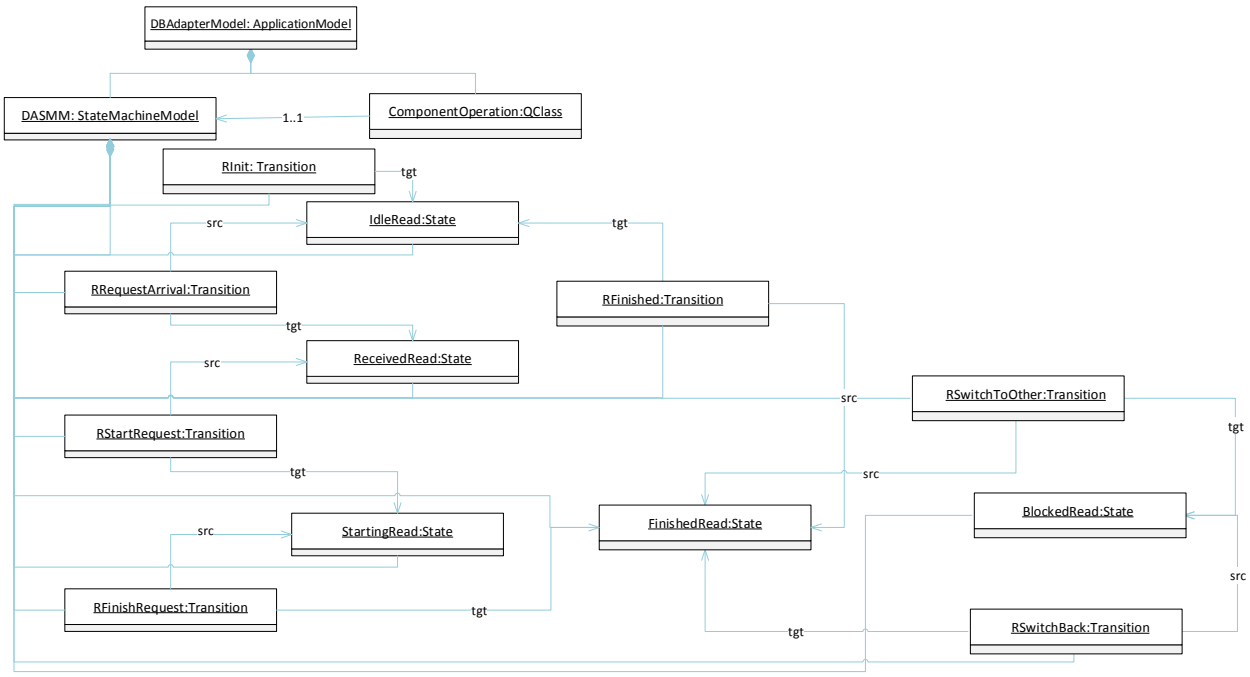

Figure A.4: Application Model for DB Adapter Component.

evident from the figure that transitions like *UFRequestArrival*, *UFStartRequest* and *UFFinishRequest* in *WebForm* application model are mapped to transitions in the *ET* context model via transition mapping objects like *RequestArrTM*, *StartRequTM* and *FinishRequTM*. Furthermore, the third equation is considered where mapping of a each pair of states in the *WebForm* application model to at least one pair in the *ET* context model are evaluated. This is to ensure that mapping within the states and transitions of *WebForm* application model and *ET* context model is consistent.

line 14 shows the validation of the *subscribe()* operation is achieved by loading the *AudioStoreModel* and *ET* context model so that the mapping between those models can be established. This mapping is established in the model mapping named *SOp2EtMapping* as shown in Fig A.10. The *ComClassMapping* controls the mapping between source and target *Component Operation* classes in application model and context model respectively. It is important to note that each *ClassMapping* should have a reference to *StateMachineModelMapping* to determine that class belongs to its own sate machine model. *StateMachineModelMapping* controls the mapping between source and target state machine model named *SSMMStateMachineModelMapping*that contains a list of all the states and transitions that belong to the classes in the *AudioRentalModel* and *RT* context model. The *StateMapping* and *Transition-*

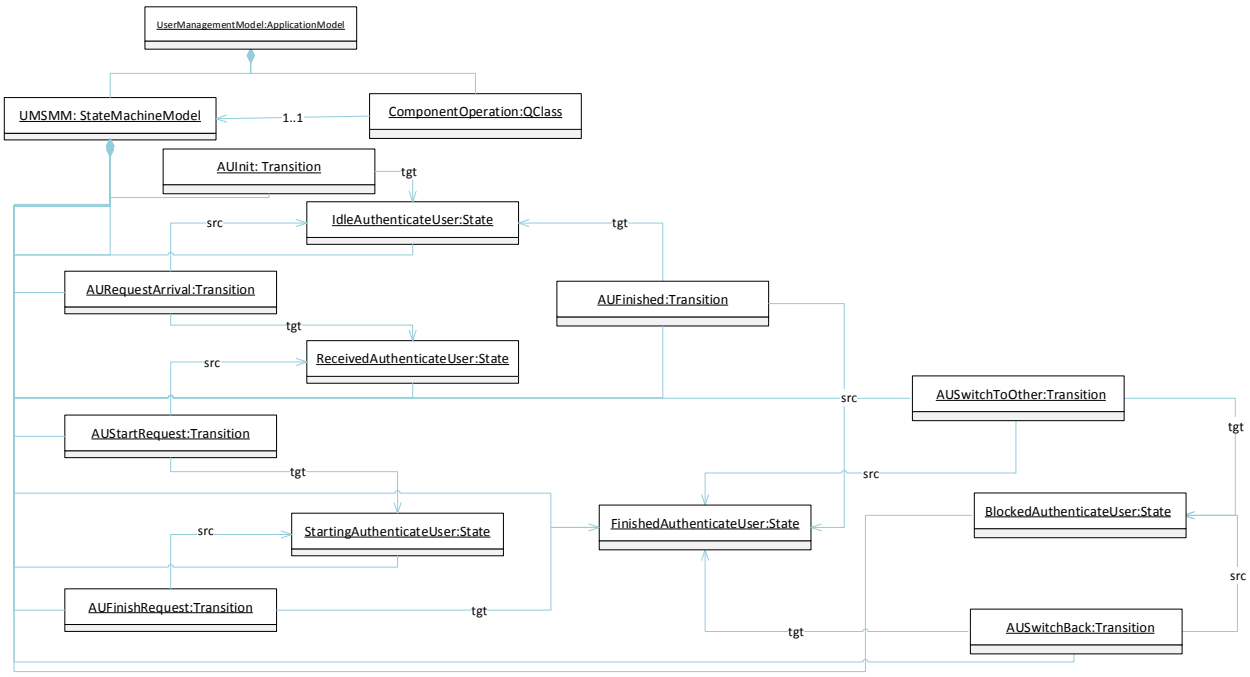

Figure A.5: Application Model for User Management Component.

Mapping control mapping of each state and transition in state machine model of the *AudioRentalModel* to a state in state machine model of the *RT* context model as required by the *StateMachineModelMapping*.

As discussed in Chapter 5,we show application of the three conditions to the *SOp2EtMapping* example of the mapping models in our case study. Fig A.10 shows set of states and transitions; and the mapping instances that connect the states and transitions between application and context models. The equation one requires that for each states in state machine model of *AudioStore* application model, there is at least one state in the *ET* context model to which it can be mapped. We can see in Fig A.10 that each of the states like *SAIdle*, *SRequestAvailable* and *SHandlingRequest* are linked to relevant states in the *ET* context model via state mapping objects like *IdleSM*, *RequestAvailSM* and *HandlingRequestSM*.

The second equation requires that for each transition in state machine model of *AudioStore* application model, there is at least one transition in the *ET* context model to which it can be mapped. It is also evident from the figure that transitions like *SARequestArrival*, *SAStartRequest* and *SAFinishRequest* in *AudioRental* application model are mapped to transitions in the *ET* context model via transition mapping

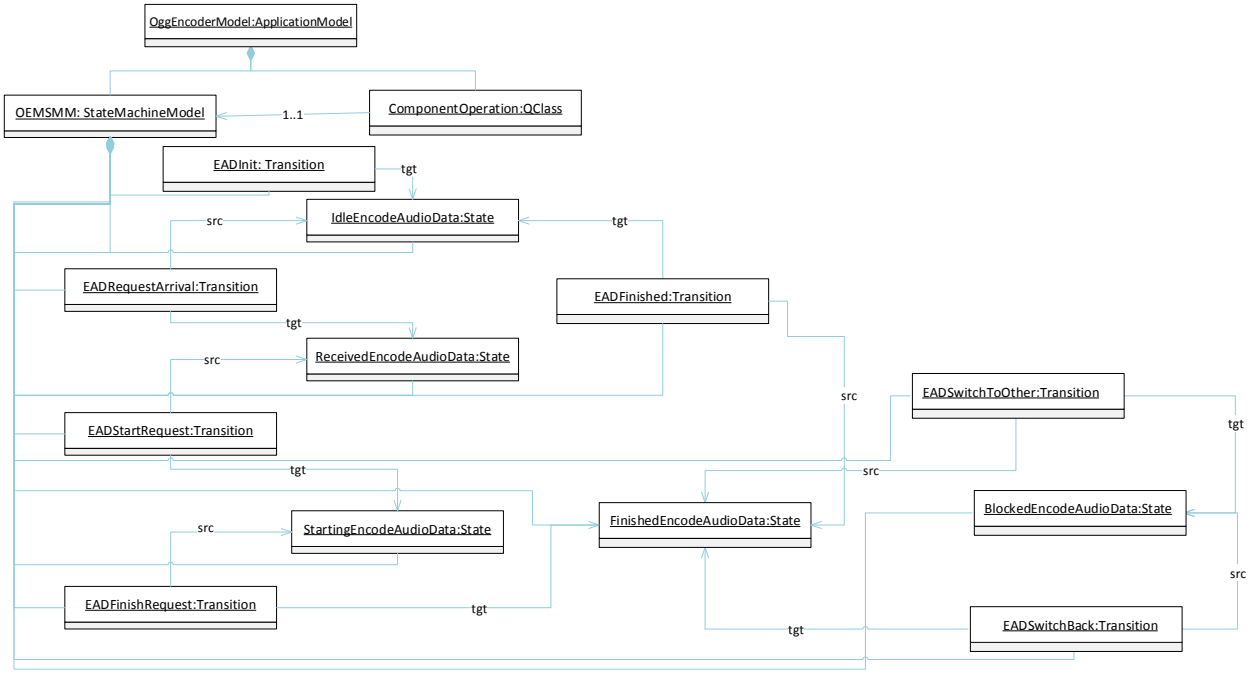

Figure A.6: Application Model for Ogg Encoder Component.

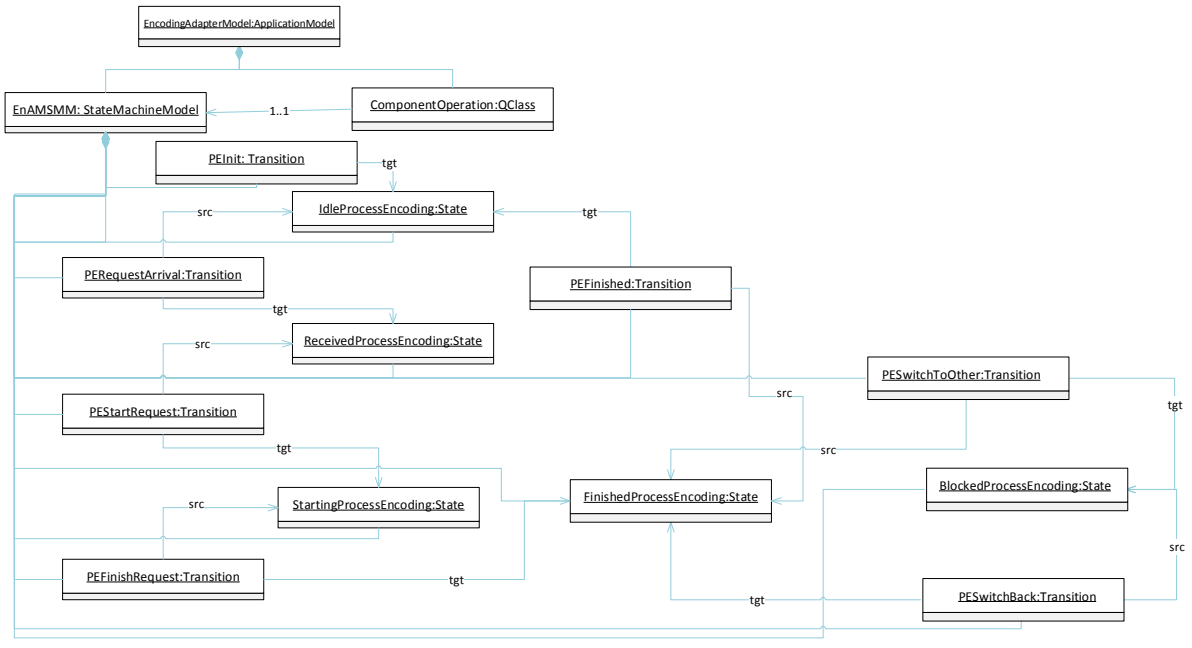

Figure A.7: Application Model for Encoding Adapter Component.

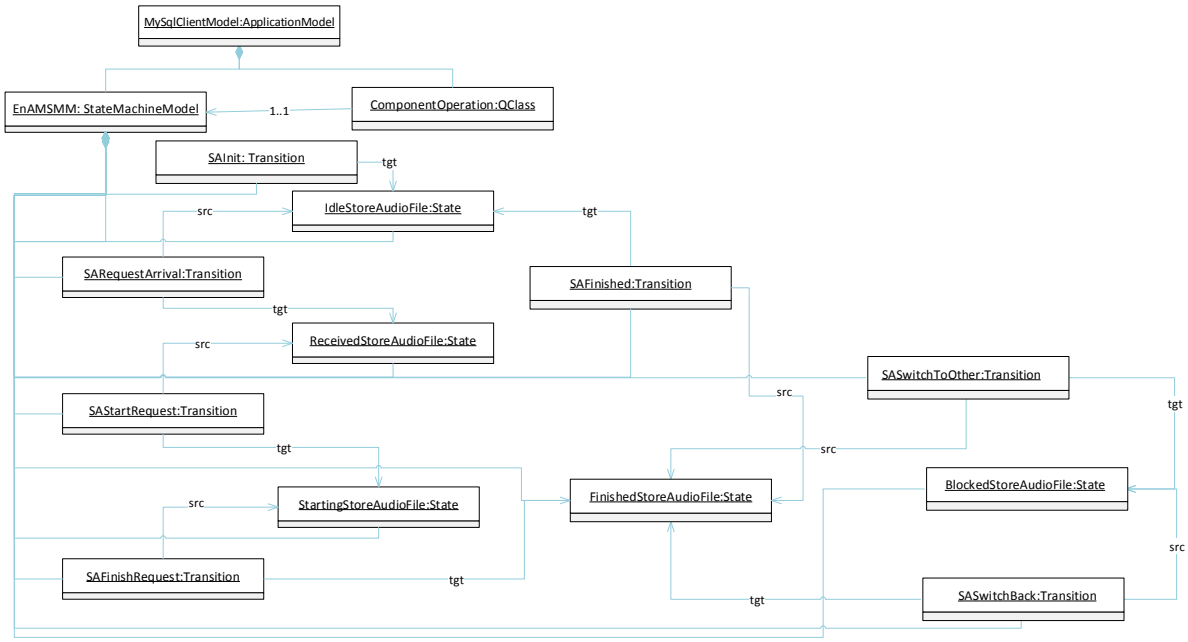

Figure A.8: Application Model for MySql Client Component.

objects like *RequestArrTM*, *StartRequTM* and *FinishRequTM*. Moreover, the third equation is considered where mapping of a each pair of states in the *AudioStore* application model to at least one pair in the *ET* context model are evaluated. This is ensure that mapping within the states and transitions of *AudioStore* application model and *ET* context model is checked to be consistent.

Validating the *read()* operation is achieved by loading the *DBAdapterModel* and *ET* context model so that the mapping between those models can be established. This mapping is established in the model mapping named *ROp2EtMapping* as shown in Fig A.11. The *COpClassMapping* controls the mapping between source and target *Component Operation* classes in application model and context model respectively. *StateMachineModelMapping* controls the mapping between source and target state machine model named *CSSMMStateMachineModelMapping*that contains a list of all the states and transitions that belong to the classes in the *DBAdapterModel* and *ET* context model. The *StateMapping* and *TransitionMapping* control mapping of each state and transition in state machine model of the *DBAdapterModel* to a state in state machine model of the *ET* context model as required by the *StateMachineModelMapping*.

Fig A.11 also shows set of states and transitions; and the mapping instances that con-

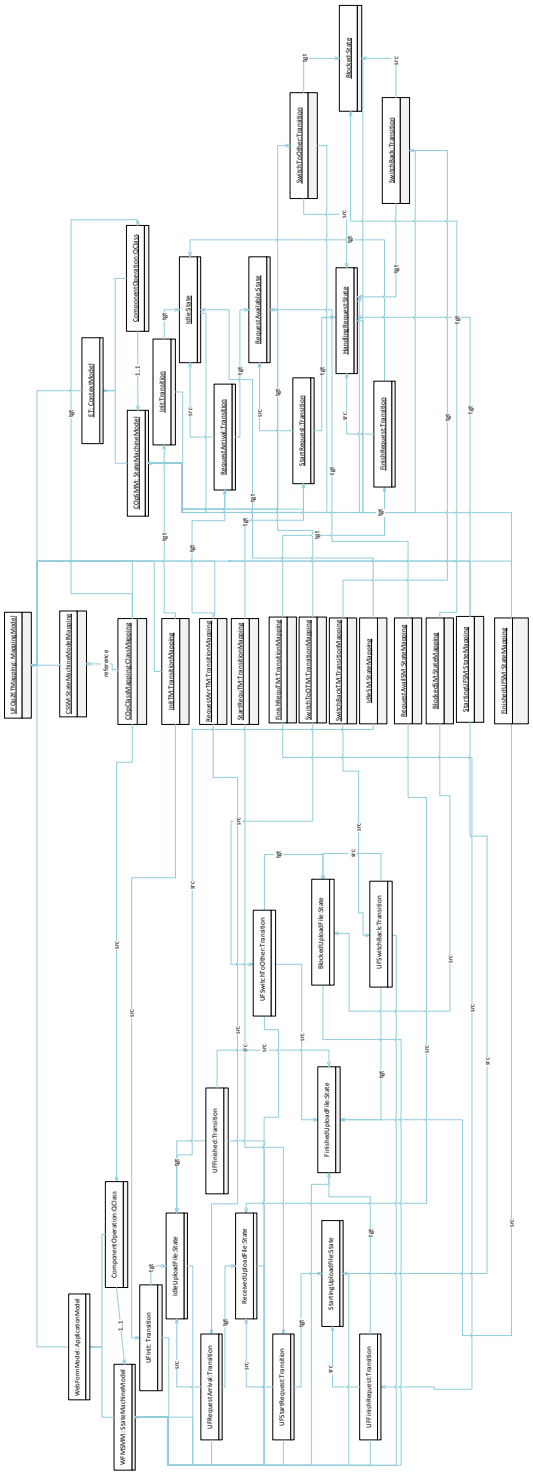

Figure A.9: *UFOp2ETMappingModel* for Validating *uploadFile* Operation.

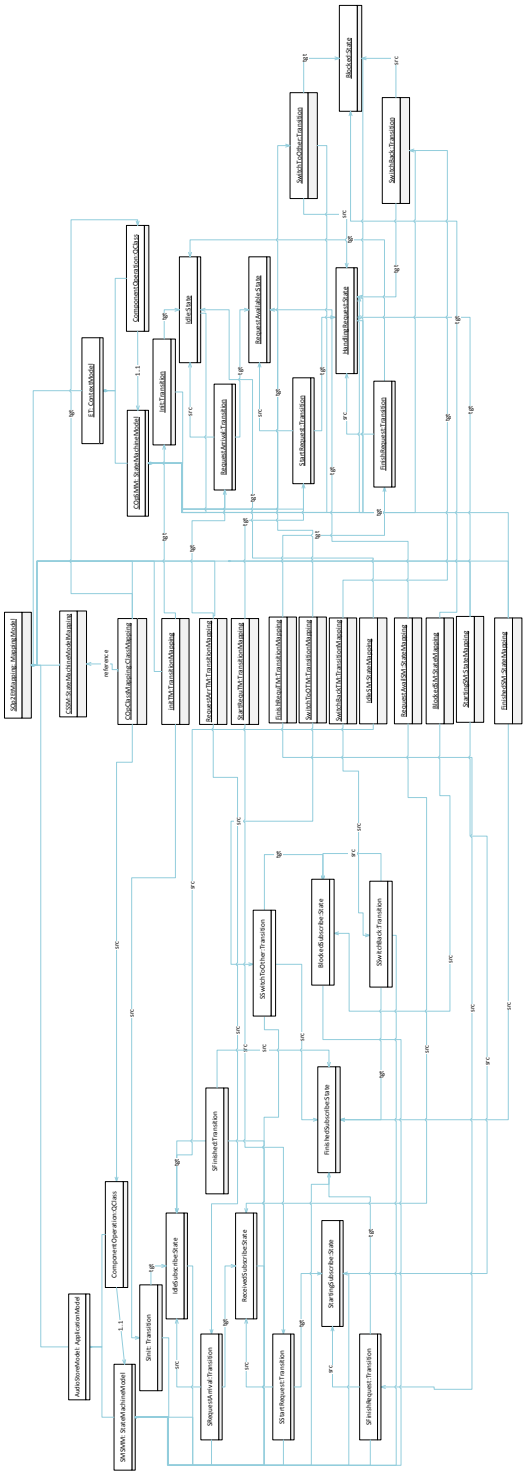

Figure A.10: *SOp2EtMapping* for Validating *subscribe* Operation.

nect the states and transitions between application and context models. The equation one requires that for each states in state machine model of *DBAdapter* application model, there is at least one state in the *ET* context model to which it can be mapped. We can see in Fig A.11 that each of the states like *RIdle*, *RRequestAvailable*, *RHandlingRequest*, *RSwitchToOther* and *RSwitchBack* are linked to relevant states in the *ET* context model via state mapping objects like *IdleSM*, *RequestAvailSM* and *HandlingRequestSM*.

As discussed in Chapter 5, we show application of the three conditions to the model mapping named *ROp2EtMapping* example of the mapping models in our case study. Fig A.11 shows set of states and transitions; and the mapping instances that connect the states and transitions between application and context models. The equation one requires that for each states in state machine model of *DBAdapter* application model, there is at least one state in the *ET* context model to which it can be mapped. We can see in Fig A.11 that each of the states like *RIdle*, *RRequestAvailable* and *RHandlingRequest* are linked to relevant states in the *ET* context model via state mapping objects like *IdleSM*, *RequestAvailSM* and *HandlingRequestSM*.

The second equation requires that for each transition in state machine model of *AudioRental* application model, there is at least one transition in the *ET* context model to which it can be mapped. It is also evident from the figure that transitions like *RRequestArrival*, *RStartRequest* and *RFinishRequest* in *DBAdapter* application model are mapped to transitions in the *ET* context model via transition mapping objects like *RequestArrTM*, *StartRequTM* and *FinishRequTM*. Moreover, the third equation is considered where mapping of a each pair of states in the *DBAdapter* application model to at least one pair in the *ET* context model are evaluated. This is ensure that mapping within the states and transitions of *DBAdapter* application model and *ET* context model is checked to be consistent.

Validating the *authenticateUser()* operation is achieved by loading the *UserManagementModel* and *ET* context model so that the mapping between those models can be established. This mapping is established in the model mapping named *AUOp2EtMapping* as shown in Fig A.12. The *COpClassMapping* controls the mapping between source and target *Component Operation* classes in application model and context model respectively. *StateMachineModelMapping* controls the mapping between source and target state machine model named *SMMStateMachineModelMapping*that contains a list of all the states and transitions that belong to the classes in the *UserManagementModel* and *ET* context model. The *StateMapping* and *TransitionMapping* control mapping of each state and transition in state machine model of the *UserManagementModel* to a state in state machine model of the *ET* context model as required by the *StateMachineModelMapping*.

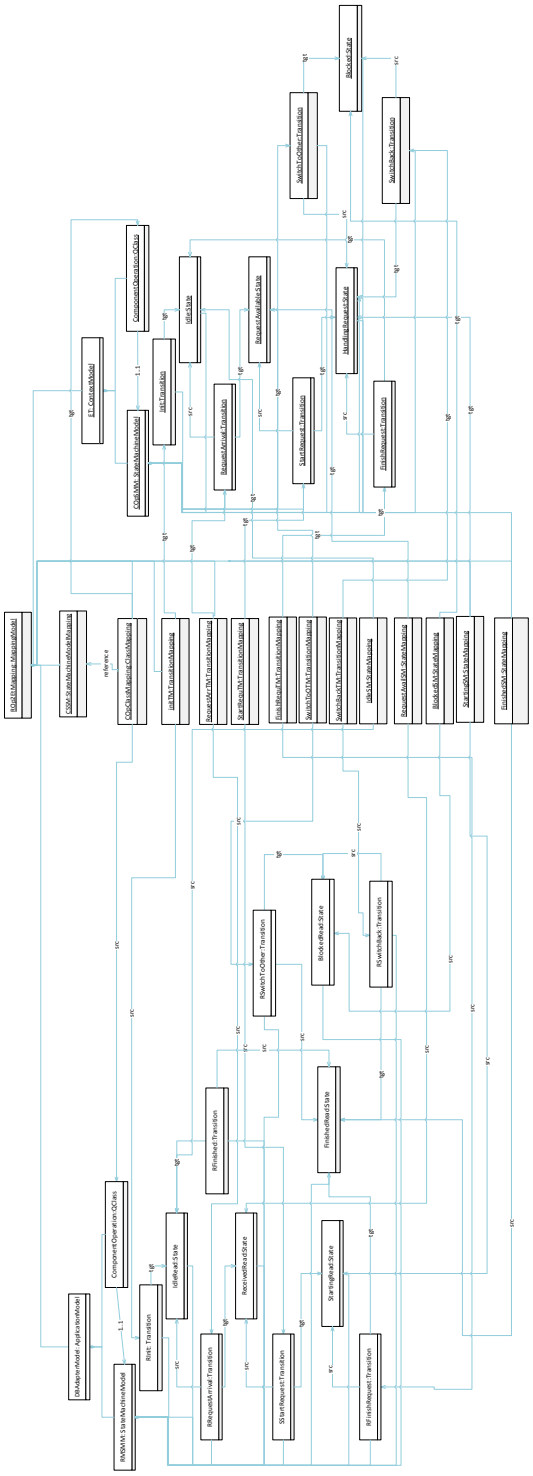

Figure A.11: *ROp2EtMapping* for Validating *read* Operation.

Fig A.12 also shows set of states and transitions; and the mapping instances that connect the states and transitions between application and context models. The equation one requires that for each states in state machine model of *UserManagement* model, there is at least one state in the *ET* context model to which it can be mapped. We can see in Fig A.12 that each of the states like *AUIdle*, *AURequestAvailable*, *AUHandlingRequest*, *AUSwitchToOther* and *AUSwitchBack* are linked to relevant states in the *ET* context model via state mapping objects like *IdleSM*, *RequestAvailSM* and *HandlingRequestSM*.

As discussed in Chapter 5, we show application of the three conditions to the model mapping named *AUOp2EtMapping* example of the mapping models in our case study. Fig A.12 shows set of states and transitions; and the mapping instances that connect the states and transitions between application and context models. The equation one requires that for each states in state machine model of *UserManagement* model, there is at least one state in the *ET* context model to which it can be mapped. We can see in Fig A.12 that each of the states like *AUIdle*, *AURequestAvailable* and *AUHandlingRequest* are linked to relevant states in the *ET* context model via state mapping objects like *IdleSM*, *RequestAvailSM* and *HandlingRequestSM*.

The second equation requires that for each transition in state machine model of *UserManagement* model, there is at least one transition in the *ET* context model to which it can be mapped. It is also evident from the figure that transitions like *AURequestArrival*, *AUStartRequest* and *AUFinishRequest* in *UserManagement* application model are mapped to transitions in the *ET* context model via transition mapping objects like *RequestArrTM*, *StartRequTM* and *FinishRequTM*. Moreover, the third equation is considered where mapping of a each pair of states in the *UserManagement* application model to at least one pair in the *ET* context model are evaluated. This is ensure that mapping within the states and transitions of *UserManagement* model and *ET* context model is checked to be consistent.

Validating the *encodeAudioData()* operation is achieved by loading the *OggEncoderModel* and *ET* context model so that the mapping between those models can be established. This mapping is established in the model mapping named *EAOp2EtMapping* as shown in Fig A.13. The *COpClassMapping* controls the mapping between source and target *Component Operation* classes in application model and context model respectively. *StateMachineModelMapping* controls the mapping between source and target state machine model named *CSSMMStateMachineModelMapping*that contains a list of all the states and transitions that belong to the classes in the *OggEncoderModel* and *ET* context model. The *StateMapping* and *TransitionMapping* control mapping of each state and transition in state machine model of the *OggEncodert-*

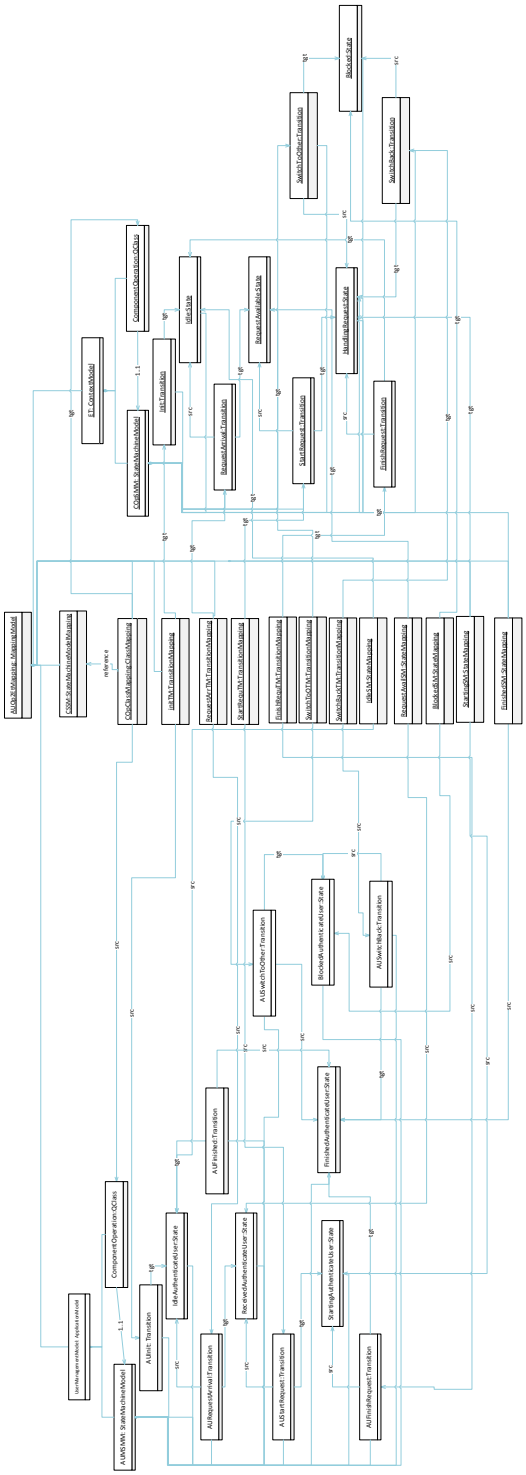

Figure A.12: *AOp2EtMapping* for Validating *authenticateUser* Operation.

Model to a state in state machine model of the *ET* context model as required by the *StateMachineModelMapping*.

Fig A.13 also shows set of states and transitions; and the mapping instances that connect the states and transitions between application and context models. The equation one requires that for each states in state machine model of *OggEncoder* model, there is at least one state in the *ET* context model to which it can be mapped. We can see in Fig A.13 that each of the states like *EAIdle*, *EARequestAvailable*, *EAHandlingRequest*, *EASwitchToOther* and *EASwitchBack* are linked to relevant states in the *ET* context model via state mapping objects like *IdleSM*, *RequestAvailSM* and *HandlingRequestSM*.

As discussed in Chapter 5, we show application of the three conditions to the model mapping named *EAOp2EtMapping* example of the mapping models in our case study. Fig A.13 shows set of states and transitions; and the mapping instances that connect the states and transitions between application and context models. The equation one requires that for each states in state machine model of *OggEncoder* model, there is at least one state in the *ET* context model to which it can be mapped. We can see in Fig A.13 that each of the states like *EAIdle*, *EARequestAvailable* and *EAHandlingRequest* are linked to relevant states in the *ET* context model via state mapping objects like *IdleSM*, *RequestAvailSM* and *HandlingRequestSM*.

The second equation requires that for each transition in state machine model of *OggEncoder* application model, there is at least one transition in the *ET* context model to which it can be mapped. It is also evident from the figure that transitions like *EARequestArrival*, *EAStartRequest* and *EAFinishRequest* in *OggEncoder* application model are mapped to transitions in the *ET* context model via transition mapping objects like *RequestArrTM*, *StartRequTM* and *FinishRequTM*. Moreover, the third equation is considered where mapping of a each pair of states in the *OggEncoder* application model to at least one pair in the *ET* context model are evaluated. This is ensure that mapping within the states and transitions of *OggEncoder* application model and *ET* context model is checked to be consistent.

Validating the *processEncoding()* operation is achieved by loading the *EncodingAdapterModel* and *ET* context model so that the mapping between those models can be established. This mapping is established in the model mapping named *PEOp2EtMapping* as shown in Fig A.14. The *COpClassMapping* controls the mapping between source and target *Component Operation* classes in application model and context model respectively. *StateMachineModelMapping* controls the mapping between source and target state machine model named *CSSMMStateMachineModelMapping*that contains a list of all the states and transitions that belong to the classes in the *OggEncodert-*

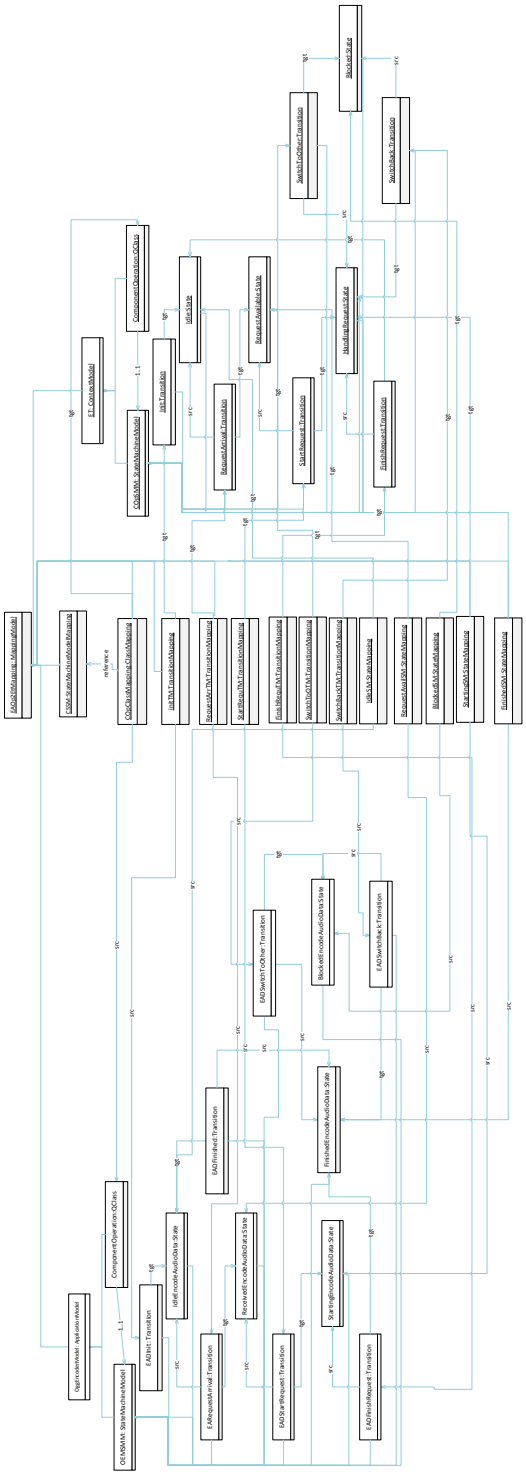

Figure A.13: *EAOp2EtMapping* for Validating *encodeAudioData* Operation.

Model and *ET* context model. The *StateMapping* and *TransitionMapping* control mapping of each state and transition in state machine model of the *OggEncodertModel* to a state in state machine model of the *ET* context model as required by the *StateMachineModelMapping*.

Fig A.14 also shows set of states and transitions; and the mapping instances that connect the states and transitions between application and context models. The equation one requires that for each states in state machine model of *EncodingAdapter* model, there is at least one state in the *ET* context model to which it can be mapped. We can see in Fig A.14 that each of the states like *PEIdle*, *PERequestAvailable*, *PEHandlingRequest*, *PESwitchToOther* and *EASwitchBack* are linked to relevant states in the *ET* context model via state mapping objects like *IdleSM*, *RequestAvailSM* and *HandlingRequestSM*.

As discussed in Chapter 5, we show application of the three conditions to the model mapping named *PEOp2EtMapping* example of the mapping models in our case study. Fig A.14 shows set of states and transitions; and the mapping instances that connect the states and transitions between application and context models. The equation one requires that for each states in state machine model of *EncodingAdapter* model, there is at least one state in the *ET* context model to which it can be mapped. We can see in Fig A.14 that each of the states like *PEIdle*, *PERequestAvailable* and *PEHandlingRequest* are linked to relevant states in the *ET* context model via state mapping objects like *IdleSM*, *RequestAvailSM* and *HandlingRequestSM*.

The second equation requires that for each transition in state machine model of *EncodingAdapter* application model, there is at least one transition in the *ET* context model to which it can be mapped. It is also evident from the figure that transitions like *PERequestArrival*, *PEStartRequest* and *PEFinishRequest* in *EncodingAdapter* application model are mapped to transitions in the *ET* context model via transition mapping objects like *RequestArrTM*, *StartRequTM* and *FinishRequTM*. Moreover, the third equation is considered where mapping of a each pair of states in the *EncodingAdapter* application model to at least one pair in the *ET* context model are evaluated. This is ensure that mapping within the states and transitions of *EncodingAdapter* application model and *ET* context model is checked to be consistent.

Validating the *storeAudioFile()* operation is achieved by loading the *MySqlClientModel* and *ET* context model so that the mapping between those models can be established. This mapping is established in the model mapping named *SAOp2EtMapping* as shown in Fig A.15. The *COpClassMapping* controls the mapping between source and target *Component Operation* classes in application model and context model respectively. *StateMachineModelMapping* controls the mapping between source and

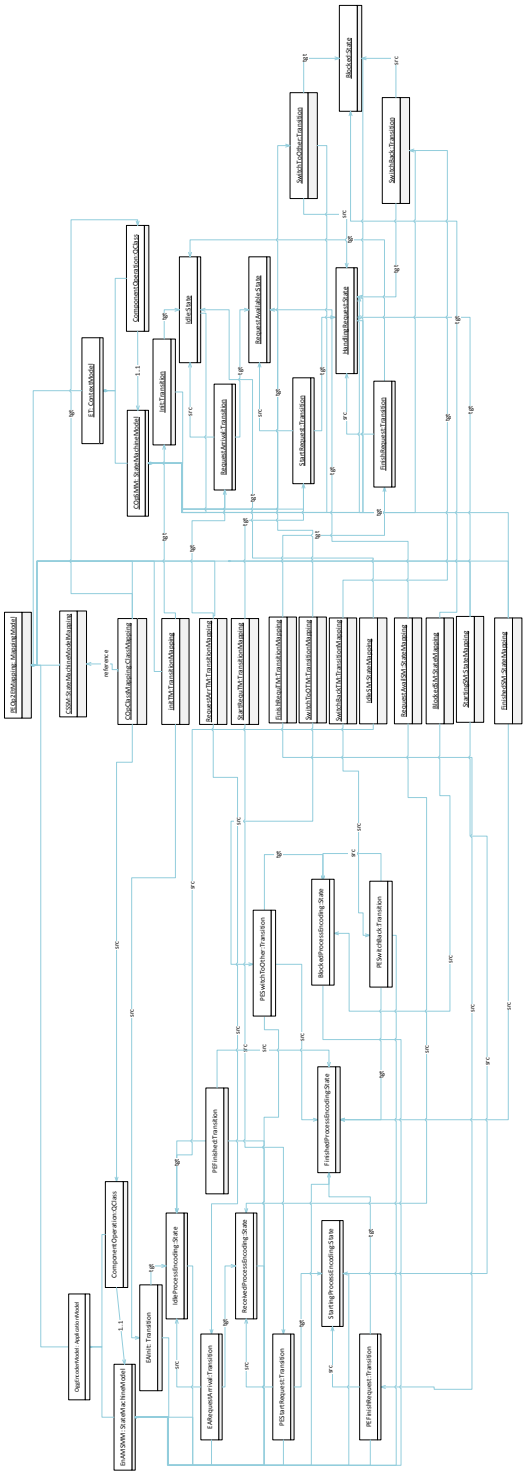

Figure A.14: *PEOp2EtMapping* for Validating *ProcessEncoding* Operation.

target state machine model named *CSSMMStateMachineModelMapping*that contains a list of all the states and transitions that belong to the classes in the *MySqlClient-Model* and *ET* context model. The *StateMapping* and *TransitionMapping* control mapping of each state and transition in state machine model of the *MySqlClient-Model* to a state in state machine model of the *ET* context model as required by the *StateMachineModelMapping*.

Fig A.15 also shows set of states and transitions; and the mapping instances that connect the states and transitions between application and context models. The equation one requires that for each states in state machine model of *MySqlClientModel* model, there is at least one state in the *ET* context model to which it can be mapped. We can see in Fig A.15 that each of the states like *SAIdle*, *SARequestAvailable*, *SAHandlingRequest*, *SASwitchToOther* and *SASwitchBack* are linked to relevant states in the *ET* context model via state mapping objects like *IdleSM*, *RequestAvailSM* and *HandlingRequestSM*.

As discussed in Chapter 5, we show application of the three conditions to the model mapping named *SAOp2EtMapping* example of the mapping models in our case study. Fig A.15 shows set of states and transitions; and the mapping instances that connect the states and transitions between application and context models. The equation one requires that for each states in state machine model of *MySqlClient* model, there is at least one state in the *ET* context model to which it can be mapped. We can see in Fig A.15 that each of the states like *SAIdle*, *SARequestAvailable* and *SAHandlingRequest* are linked to relevant states in the *ET* context model via state mapping objects like *IdleSM*, *RequestAvailSM* and *HandlingRequestSM*.

The second equation requires that for each transition in state machine model of *MySqlClient* application model, there is at least one transition in the *ET* context model to which it can be mapped. It is also evident from the figure that transitions like *SARequestArrival*, *SAStartRequest* and *SAFinishRequest* in *MySqlClient* application model are mapped to transitions in the *ET* context model via transition mapping objects like *RequestArrTM*, *StartRequTM* and *FinishRequTM*. Moreover, the third equation is considered where mapping of a each pair of states in the *MySqlClient* application model to at least one pair in the *ET* context model are evaluated. This is ensure that mapping within the states and transitions of *MySqlClient* application model and *ET* context model is checked to be consistent.

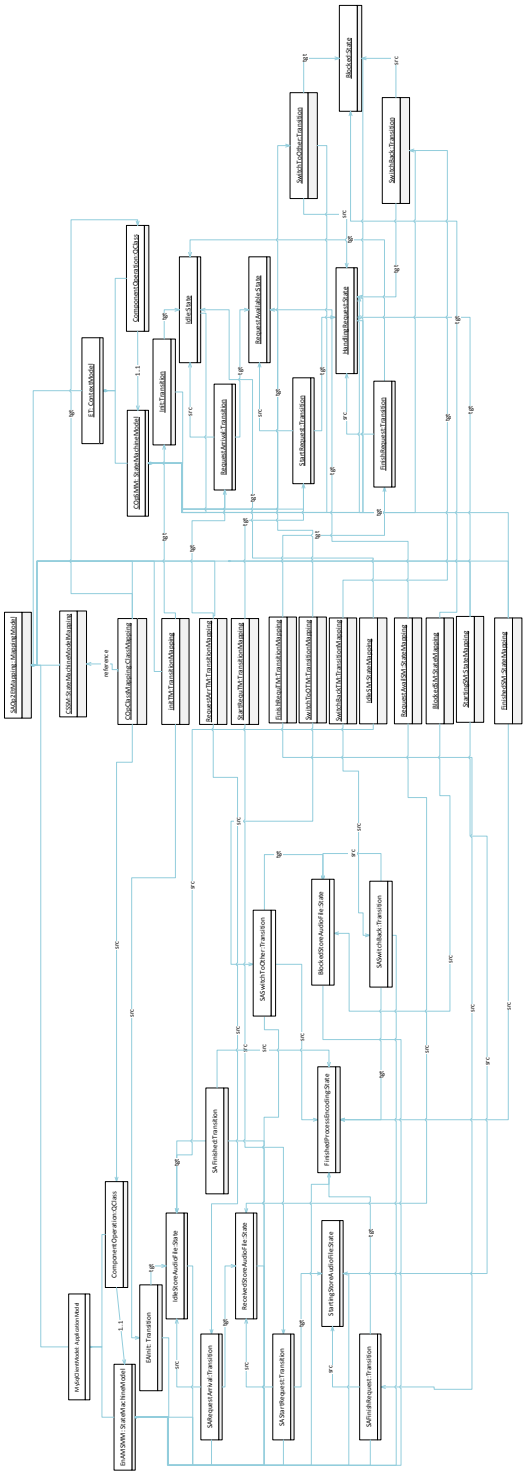

Figure A.15: *SAOp2EtMapping* for Validating *storeAudioFile* Operation.

A.2 TLA+ Specification Generated

Listing A.3 shows the TLA+ specification of *Delta Time* module. It is equivalent to QML/CS specifications of the *Delta Time* discussed in Listing A.1. Lines 2-9 extend *RealTime* module and make use of the module of a *service*. The *RealTime* specification is originally introduced by Abadi and Lamport in [4], which allows the definition of the variable *now*. It helps specifying the constraints over variable *now* to indicate that the time can only move forward. Lines 11-20 show the specification of transitions *Init* and *OnRequestArrival*. The variables *StartDelta* and *LastDeltaTime* represent the start time of the last request and the time between the last two requests, respectively.

```
1 ---------- MODULE DeltaTime ----------
2 EXTENDS RealTime
3
4 VARIABLES unhandledRequest, inState
5 ----------------------------------------
6 VARIABLES  DeltaTime, hadOpCall, StartDelta, LastDeltaTime
7 ----------------------------------------
8
9 op == INSTANCE InRT
10
11 OnInit == op!RequestArrival =>
12                                 /\ DeltaTime \in Real
13                                 /\ DeltaTime=0
14                                 /\ StartDelta=now
15 OnRequestArrival == op!RequestArrival =>
16                                         /\ LastDeltaTime'=now
17                                         /\ LastDeltaTime'= now -
       StartDelta
18                                         /\ StartDelta'=now
19 Spec == /\ op!InRT
20         /\ [][OnInit /\ OnRequestArrival]_(DeltaTime)
21 ========================================
```

Listing A.3: The TLA+ Specification for Delta time.

Listing A.4 shows the TLA+ specification of *Data Rate* module. It is equivalent to QML/CS specification of the *Data Rate* discussed in Listing A.2. Lines 1-8 represent the measurement *Data Rate* that extends *Real Time* module, makes use of the module of a *component*. Lines 26-28 show specifying variable *LastInterval* that defines the interval between two successive data emissions. The *Spec* in both listings show the state machine specification with a list of states that are part of state machine model of the measurement context. The details of each state are described in their relevant

context model and their concrete definition is provided in this specification.

```
1 ---------- MODULE DataRate ----------
2 EXTENDS RealTime
3
4 VARIABLES unhandledRequest, inState
5 ------------------------------------------------
6 VARIABLES  LastInterval, hadOpCall, AccInterval, Start
7 ------------------------------------------------
8
9 op == INSTANCE DR
10
11 OnInit == op!Init =>
12                     /\ LastInterval \in Real
13                     /\ LastInterval=0
14                     /\ AccInterval=0
15 OnStartWorking == op!StartWorking =>
16                                     /\ Start=now
17                                     /\ AccInterval'=0
18 OnGetBlocked == op!GetBlocked =>
19                                   /\ AccInterval'= AccInterval + now -
    Start
20 OnGetUnBlocked == op!GetUnBlocked =>
21                                     /\ Start'=now
22 OnDoSendData == op!DoSendData =>
23                                   /\ LastInterval'= LastInterval + now -
    Start
24                                   /\ Start'=now
25                                   /\ AccInterval'=0
26 Spec == /\ op!DR
27         /\ [][OnInit/\OnStartWorking/\OnGetBlocked/\OnGetUnBlocked/\
    OnDoSendData]_(DataRate)
28 ================================
```

Listing A.4: The TLA+ Specification for Data Rate.

A.2.1 TLA+ Specification of Application Interface

Listing A.5 shows the TLA+ specification of a global representation of a state for number of components. Also, it shows abstract actions that are defined via a Boolean constant.

```
1 ---- MODULE AudioRentalInterface ----
2
3  (*****************************************)
```

```
4   (*Representation of the AudioRentalŠs state.*)
5   (*****************************************)
6
7  VARIABLE AudioRentalState
8
9  CONSTANT rentAudio (_,_)
10 CONSTANT InitialAudioRentalStates
11
12 CONSTANT SendData(_,_,_)
13
14  (*********************************************************)
15  (*An abstract action that is defined via a Boolean constant.*)
16  (*********************************************************)
17
18 ASSUME \A v, AudioRentalStOld, AudioRentalStNew:
19    /\ rentAudio (AudioRentalStOld, AudioRentalStNew) \in BOOLEAN
20    /\ SendData(v, AudioRentalStOld, AudioRentalStNew) \in BOOLEAN
21
22 ====
23 ---- MODULE WebFormInterface ----
24
25  (*************************************)
26  (*Representation of the WebFormŠs state.*)
27  (*************************************)
28
29 VARIABLE WebFormState
30
31 CONSTANT uploadFile (_,_)
32 CONSTANT InitialWebFormStates
33
34 CONSTANT SendData(_,_,_)
35
36  (*********************************************************)
37  (*An abstract action that is defined via a Boolean constant.*)
38  (*********************************************************)
39
40 ASSUME \A v, WebFormStOld, WebFormStNew:
41    /\ uploadFile (WebFormStOld, WebFormStNew) \in BOOLEAN
42    /\ SendData(v, WebFormStOld, WebFormStNew) \in BOOLEAN
43
44 ====
45 ---- MODULE AudioStoreInterface ----
46
47  (*****************************************)
48  (*Representation of the AudioStoreŠs state.*)
49  (*****************************************)
50
51 VARIABLE AudioStoreState
52
```

```
53 CONSTANT subscribe (_,_)
54 CONSTANT InitialAudioStoreStates
55
56 CONSTANT SendData(_,_,_)
57
58  (*************************************************************)
59  (*An abstract action that is defined via a Boolean constant.*)
60  (*************************************************************)
61
62 ASSUME \A v, AudioStoreStOld, AudioStoreStNew:
63   /\ subscribe (AudioStoreStOld, AudioStoreStNew) \in BOOLEAN
64   /\ SendData(v, AudioStoreStOld, AudioStoreStNew) \in BOOLEAN
65
66 ====
67 ---- MODULE DBAdapterInterface ----
68
69  (*****************************************)
70  (* Representation of the DBAdapterŠs state.*)
71  (*****************************************)
72
73 VARIABLE DBAdapterState
74
75 CONSTANT Read (_,_)
76 CONSTANT InitialDBAdapterStates
77
78 CONSTANT SendData(_,_,_)
79
80  (*************************************************************)
81  (*An abstract action that is defined via a Boolean constant.*)
82  (*************************************************************)
83
84 ASSUME \A v, DBAdapterStOld, DBAdapterStNew:
85   /\ Read (DBAdapterStOld, DBAdapterStNew) \in BOOLEAN
86   /\ SendData(v, DBAdapterStOld, DBAdapterStNew) \in BOOLEAN
87 ====
88 ---- MODULE UserManagmentInterface ----
89
90  (*****************************************)
91  (*Representation of the UserManagmentŠs state.*)
92  (*****************************************)
93
94 VARIABLE UserManagmentState
95
96 CONSTANT authenticateUser (_,_)
97 CONSTANT InitialUserManagmentStates
98
99 CONSTANT SendData(_,_,_)
100
101  (*************************************************************)
```

```
102  (*An abstract action that is defined via a Boolean constant.*)
103  (*********************************************************)
104
105 ASSUME \A v, UserManagmentStOld, UserManagmentStNew:
106    /\ authenticateUser (UserManagmentStOld, UserManagmentStNew) \in
       BOOLEAN
107    /\ SendData(v, UserManagmentStOld, UserManagmentStNew) \in BOOLEAN
108 ====
109 ---- MODULE OggEncoderInterface ----
110
111  (*****************************************)
112  (* Representation of the OggEncoderŠs state.*)
113  (*****************************************)
114
115 VARIABLE OggEncoderState
116
117 CONSTANT encodeAudioData (_,_)
118 CONSTANT InitialOggEncoderStates
119
120 CONSTANT SendData(_,_,_)
121
122  (*********************************************************)
123  (*An abstract action that is defined via a Boolean constant.*)
124  (*********************************************************)
125
126 ASSUME \A v, OggEncoderStOld, OggEncoderStNew:
127    /\ encodeAudioData (OggEncoderStOld, OggEncoderStNew) \in BOOLEAN
128    /\ SendData(v, OggEncoderStOld, OggEncoderStNew) \in BOOLEAN
129 ====
130 ---- MODULE EncodingAdapterInterface ----
131
132  (*********************************************)
133  (* Representation of the EncodingAdapterŠs state.*)
134  (*********************************************)
135
136 VARIABLE EncodingAdapterState
137
138 CONSTANT processEncoding (_,_)
139 CONSTANT InitialEncodingAdapterStates
140
141 CONSTANT SendData(_,_,_)
142
143  (*********************************************************)
144  (*An abstract action that is defined via a Boolean constant.*)
145  (*********************************************************)
146
147 ASSUME \A v, EncodingAdapterStOld, EncodingAdapterStNew:
148    /\ processEncoding (EncodingAdapterStOld, EncodingAdapterStNew) \in
       BOOLEAN
```

```
149   /\ SendData(v, EncodingAdapterStOld, EncodingAdapterStNew) \in BOOLEAN
150 ====
151 ---- MODULE MySqlClientInterface ----
152  (*****************************************)
153  (* Representation of the MySqlClientŠs state.*)
154  (*****************************************)
155
156 VARIABLE MySqlClientState
157
158 CONSTANT authenticateUser (_,_)
159 CONSTANT storeAudioFile (_,_)
160 CONSTANT getUserAudioSubscriptions (_,_)
161 CONSTANT loadAudioFile (_,_)
162 CONSTANT InitialMySqlClientStates
163
164 CONSTANT SendData(_,_,_)
165
166  (*********************************************************)
167  (*An abstract action that is defined via a Boolean constant.*)
168  (*********************************************************)
169
170 ASSUME \A v, MySqlClientStOld, MySqlClientStNew:
171   /\ authenticateUser (MySqlClientStOld, MySqlClientStNew) \in BOOLEAN
172   /\ storeAudioFile (MySqlClientStOld, MySqlClientStNew) \in BOOLEAN
173   /\ getUserAudioSubscriptions (MySqlClientStOld, MySqlClientStNew) \in
      BOOLEAN
174   /\ loadAudioFile (MySqlClientStOld, MySqlClientStNew) \in BOOLEAN
175   /\ SendData(v, MySqlClientStOld, MySqlClientStNew) \in BOOLEAN
176 ====
```

Listing A.5: The TLA+ Specification for Application Interface.

A.2.2 TLA+ Specification of Application

Listing A.6 shows the TLA+ specification of *Web Audio Store* application that has a number of components. Every module starts with extending its own interface as in shown line 2. For example, the *AudioRental* extends the *AudioRentalInterface*. This *AudioRentalInterface* module provides the definition of *AudioRental* interface. *AudioRentalInterface* module is only used as a helper module so that in *AudioRental* module (as shown in listing A.6) can hide its implementation. Lines 4 and 5 show that a *AudioRental* module has a number of variables. Lines 7 thru 27 demonstrate how the*AudioRentalInterface* module and its abstract actions are used to express the interactions with the environment in a *AudioRental* module.

```
1 ---- MODULE AudioRentalApp ----
2 EXTENDS AudioRentalInterface, Naturals
3
4 VARIABLE internalAudioRental
5 VARIABLE doHandle
6
7 Init == /\ internalAudioRental = 0
8         /\ doHandle = 0
9         /\ AudioRentalState \in InitialAudioRentalStates
10
11 ReceiverentAudio == /\ rentAudio(AudioRentalState, AudioRentalState')
12                     /\ doHandle = 0
13                     /\ doHandle'= 1
14                     /\ UNCHANGED internalAudioRental
15
16 HandlerentAudio == /\ doHandle = 1
17                    /\ doHandle'= 2
18                    /\ UNCHANGED <<internalAudioRental, AudioRentalState
    >>
19
20 ReplyStep == /\ doHandle = 2
21              /\ doHandle'= 0
22              /\ SendData <<internalAudioRental, AudioRentalState,
    AudioRentalState'>>
23              /\ UNCHANGED internalAudioRental
24
25 Next == \/ ReceiverentAudio
26         \/ HandlerentAudio
27         \/ ReplyStep
28
29 vars == <<AudioRentalState, internalAudioRental, doHandle>>
30
31 Spec == /\ Init
32         /\ [Next]_vars
33
34 rentAudio, []response_time < 20
35
36 VARIABLES (response_time, rentAudio.AudioRental)unhandledRequest, (
    response_time, rentAudio.AudioRental)inState
37 VARIABLES (response_time, rentAudio.AudioRental)start, (response_time,
    rentAudio.AudioRental)end, (response_time, rentAudio.AudioRental)
    inCall
38
39 (response_time, rentAudio.AudioRental) Spec == INSTANCE response_time
40 (response_time, rentAudio.AudioRental) RAOp2RtMapping.Mapping1 == [][
    response_time, rentAudio]_MM
41
42
43 Spec == /\ rentAudio
```

```
44          /\ rentAudio, response_time < 20
45          /\ (response_time, rentAudio.AudioRental)RAOp2RtMapping.Mapping1
46          /\ (response_time, rentAudio.AudioRental)Spec!Spec
47          /\ [][]response_time <20]_p
48 =====
49 ---- MODULE WebFormApp ----
50 EXTENDS WebFormInterface, Naturals
51
52 VARIABLE internalWebForm
53 VARIABLE doHandle
54
55 Init == /\ internalWebForm = 0
56          /\ doHandle = 0
57          /\ WebFormState \in InitialWebFormStates
58
59 ReceiveuploadFile == /\ uploadFile(WebFormState, WebFormState')
60                      /\ doHandle = 0
61                      /\ doHandle'= 1
62                      /\ UNCHANGED internalWebForm
63
64 HandleuploadFile == /\ doHandle = 1
65                     /\ doHandle'= 2
66                     /\ UNCHANGED <<internalWebForm, WebFormState>>
67
68 ReplyStep == /\ doHandle = 2
69              /\ doHandle'= 0
70              /\ SendData <<internalWebForm, WebFormState, WebFormState
    '>>
71              /\ UNCHANGED internalWebForm
72
73 Next == \/ ReceiveuploadFile
74         \/ HandleuploadFile
75         \/ ReplyStep
76
77 vars == <<WebFormState, internalWebForm, doHandle>>
78
79 Spec == /\ Init
80         /\ [Next]_vars
81
82 uploadFile, []execution_time < 20
83
84 VARIABLES (execution_time, uploadFile.WebForm)unhandledRequest, (
    execution_time, uploadFile.execution_time)inState
85 VARIABLES (execution_time, uploadFile.WebForm)start, (execution_time,
    uploadFile.execution_time)end, (execution_time, uploadFile.WebForm)
    inCall
86
87 (execution_time, uploadFile.WebForm) Spec == INSTANCE execution_time
```

```
88 (execution_time, uploadFile.WebForm) UFOp2EEMapping.Mapping1 == [][
       execution_time, uploadFile]_MM
89
90
91 Spec == /\ uploadFile
92         /\ uploadFile, execution_time < 20
93         /\ (execution_time, uploadFile.WebForm)UFOp2EEMapping.Mapping1
94         /\ (execution_time, uploadFile.WebForm)Spec!Spec
95         /\ [][]execution_time <20]_p
96
97 =====
98 ---- MODULE AudioStoreApp ----
99 EXTENDS AudioStoreInterface, Naturals
100
101 VARIABLE internalAudioStore
102 VARIABLE doHandle
103
104 Init == /\ internalAudioStore = 0
105         /\ doHandle = 0
106         /\ AudioStoreState \in InitialAudioStoreStates
107
108 Receivesubscribe == /\ subscribe(AudioStoreState, AudioStoreState')
109                     /\ doHandle = 0
110                     /\ doHandle'= 1
111                     /\ UNCHANGED internalAudioStore
112
113 Handlesubscribe == /\ doHandle = 1
114                    /\ doHandle'= 2
115                    /\ UNCHANGED <<internalAudioStore, AudioStoreState>>
116
117 ReplyStep == /\ doHandle = 2
118              /\ doHandle'= 0
119              /\ SendData <<internalAudioStore, AudioStoreState,
       AudioStoreState'>>
120              /\ UNCHANGED internalAudioStore
121
122 Next == \/ Receivesubscribe
123         \/ Handlesubscribe
124         \/ ReplyStep
125
126 vars == <<AudioStoreState, internalAudioStore, doHandle>>
127
128 Spec == /\ Init
129         /\ [Next]_vars
130
131 subscribe, []execution_time < 20
132
133 VARIABLES (execution_time, subscribe.AudioStore)unhandledRequest, (
       execution_time, subscribe.execution_time)inState
```

```
134 VARIABLES (execution_time, subscribe.AudioStore)start, (execution_time,
        subscribe.execution_time)end, (execution_time, subscribe.AudioStore)
        inCall
135
136 (execution_time, subscribe.AudioStore) Spec == INSTANCE execution_time
137 (execution_time, subscribe.AudioStore) SOp2EtMapping.Mapping1 == [][
        execution_time, subscribe]_MM
138
139
140 Spec == /\ subscribe
141         /\ subscribe, execution_time < 20
142         /\ (execution_time, subscribe.AudioStore)SOp2EtMapping.Mapping1
143         /\ (execution_time, subscribe.AudioStore)Spec!Spec
144         /\ [][]execution_time <20]_p
145
146 =====
147 ---- MODULE DBAdapterApp ----
148 EXTENDS DBAdapterInterface, Naturals
149
150 VARIABLE internalDBAdapter
151 VARIABLE doHandle
152
153 Init == /\ internalDBAdapter = 0
154         /\ doHandle = 0
155         /\ DBAdapterState \in InitialDBAdapterStates
156
157
158 ReceiveRead == /\ Read(DBAdapterState, DBAdapterState')
159                /\ doHandle = 0
160                /\ doHandle'= 1
161                /\ UNCHANGED internalDBAdapter
162
163 HandleRead == /\ doHandle = 1
164               /\ doHandle'= 2
165               /\ UNCHANGED <<internalDBAdapter, DBAdapterState>>
166
167 ReplyStep == /\ doHandle = 2
168              /\ doHandle'= 0
169              /\ SendData <<internalDBAdapter, DBAdapterState,
        DBAdapterState'>>
170              /\ UNCHANGED internalDBAdapter
171
172 Next == \/ ReceiveRead
173         \/ HandleRead
174         \/ ReplyStep
175
176 vars == <<DBAdapterState, internalDBAdapter, doHandle>>
177
178 Spec == /\ Init
```

```
179         /\ [Next]_vars
180
181 read, []execution_time < 40
182
183 VARIABLES (execution_time, read.DBAdapter)unhandledRequest, (
    execution_time, read.execution_time)inState
184 VARIABLES (execution_time, read.DBAdapter)start, (execution_time, read.
    execution_time)end, (execution_time, read.DBAdapter)inCall
185
186 (execution_time, read.DBAdapter) Spec == INSTANCE execution_time
187 (execution_time, read.DBAdapter) ROp2EtMapping.Mapping1 == [][
    execution_time, read]_MM
188
189
190 Spec == /\ read
191         /\ read, execution_time < 40
192         /\ (execution_time, read.DBAdapter)ROp2EtMapping.Mapping1
193         /\ (execution_time, read.DBAdapter)Spec!Spec
194         /\ [][]execution_time <40]_p
195
196 ====
197 ---- MODULE UserManagmentApp ----
198 EXTENDS UserManagmentInterface, Naturals
199
200 VARIABLE internalUserManagment
201 VARIABLE doHandle
202
203 Init == /\ internalUserManagment = 0
204         /\ doHandle = 0
205         /\ UserManagmentState \in InitialUserManagmentStates
206
207 ReceiveauthenticateUser == /\ authenticateUser(UserManagmentState,
    UserManagmentState')
208                            /\ doHandle = 0
209                            /\ doHandle'= 1
210                            /\ UNCHANGED internalUserManagment
211
212 HandleauthenticateUser == /\ doHandle = 1
213                           /\ doHandle'= 2
214                           /\ UNCHANGED <<internalUserManagment,
    UserManagmentState>>
215
216 ReplyStep == /\ doHandle = 2
217              /\ doHandle'= 0
218              /\ SendData <<internalUserManagment, UserManagmentState,
    UserManagmentState'>>
219              /\ UNCHANGED internalUserManagment
220
221 Next == \/ ReceiveauthenticateUser
```

```
222         \/ HandleauthenticateUser
223         \/ ReplyStep
224
225 vars == <<UserManagmentState, internalUserManagment, doHandle>>
226
227 Spec == /\ Init
228         /\ [Next]_vars
229
230 authenticateUser, []execution_time <= 30
231
232 VARIABLES (execution_time, authenticateUser.UserManagment)
    unhandledRequest, (execution_time, authenticateUser.execution_time)
    inState
233 VARIABLES (execution_time, authenticateUser.UserManagment)start, (
    execution_time, authenticateUser.execution_time)end, (execution_time
    , authenticateUser.UserManagment)inCall
234
235 (execution_time, authenticateUser.UserManagment) Spec == INSTANCE
    execution_time
236 (execution_time, authenticateUser.UserManagment) AUOp2EtMapping.Mapping1
     == [][execution_time, authenticateUser]_MM
237
238 Spec == /\ authenticateUser
239         /\ authenticateUser, execution_time <= 30
240         /\ (execution_time, authenticateUser.UserManagment)
    AUOp2EtMapping.Mapping1
241         /\ (execution_time, authenticateUser.UserManagment)Spec!Spec
242         /\ [][]execution_time <=30]_p
243
244 =====
245 ---- MODULE OggEncoderApp ----
246 EXTENDS OggEncoderInterface, Naturals
247
248 VARIABLE internalOggEncoder
249 VARIABLE doHandle
250
251 Init == /\ internalOggEncoder = 0
252         /\ doHandle = 0
253         /\ OggEncoderState \in InitialOggEncoderStates
254
255 ReceiveencodeAudioData == /\ encodeAudioData(OggEncoderState,
    OggEncoderState')
256                           /\ doHandle = 0
257                           /\ doHandle'= 1
258                           /\ UNCHANGED internalOggEncoder
259
260 HandleencodeAudioData == /\ doHandle = 1
261                          /\ doHandle'= 2
```

```
262                             /\ UNCHANGED <<internalOggEncoder,
    OggEncoderState>>
263
264 ReplyStep == /\ doHandle = 2
265               /\ doHandle'= 0
266               /\ SendData <<internalOggEncoder, OggEncoderState,
    OggEncoderState'>>
267               /\ UNCHANGED internalOggEncoder
268
269 Next == \/ ReceiveencodeAudioData
270         \/ HandleencodeAudioData
271         \/ ReplyStep
272
273 vars == <<OggEncoderState, internalOggEncoder, doHandle>>
274
275 Spec == /\ Init
276         /\ [Next]_vars
277
278 encodeAudioData, []execution_time <= 30
279
280 VARIABLES (execution_time, encodeAudioData.OggEncoder)unhandledRequest,
    (execution_time, encodeAudioData.execution_time)inState
281 VARIABLES (execution_time, encodeAudioData.OggEncoder)start, (
    execution_time, encodeAudioData.execution_time)end,
282           (execution_time, encodeAudioData.OggEncoder)inCall
283
284 (execution_time, encodeAudioData.OggEncoder) Spec == INSTANCE
    execution_time
285 (execution_time, encodeAudioData.OggEncoder) EAOp2EtMapping.Mapping1 ==
    [][execution_time, encodeAudioData]_MM
286
287
288 Spec == /\ encodeAudioData
289         /\ encodeAudioData, execution_time <= 30
290         /\ (execution_time, encodeAudioData.OggEncoder)EAOp2EtMapping.
    Mapping1
291         /\ (execution_time, encodeAudioData.OggEncoder)Spec!Spec
292         /\ [][]execution_time <=30]_p
293
294 =====
295 ---- MODULE EncodingAdapterApp ----
296 EXTENDS EncodingAdapterInterface, Naturals
297
298 VARIABLE internalEncodingAdapter
299 VARIABLE doHandle
300
301 Init == /\ internalEncodingAdapter = 0
302         /\ doHandle = 0
303         /\ EncodingAdapterState \in InitialEncodingAdapterStates
```

```
304
305
306 ReceiveprocessEncoding == /\ processEncoding(EncodingAdapterState,
    EncodingAdapterState')
307                           /\ doHandle = 0
308                           /\ doHandle'= 1
309                           /\ UNCHANGED internalEncodingAdapter
310
311 HandleprocessEncoding == /\ doHandle = 1
312                          /\ doHandle'= 2
313                           /\ UNCHANGED <<internalEncodingAdapter,
    EncodingAdapterState>>
314
315 ReplyStep == /\ doHandle = 2
316              /\ doHandle'= 0
317              /\ SendData <<internalEncodingAdapter, EncodingAdapterState
    , EncodingAdapterState'>>
318              /\ UNCHANGED internalEncodingAdapter
319
320 Next == \/ ReceiveprocessEncoding
321         \/ HandleprocessEncoding
322         \/ ReplyStep
323
324 vars == <<EncodingAdapterState, internalEncodingAdapter, doHandle>>
325
326 Spec == /\ Init
327         /\ [Next]_vars
328
329 processEncoding, []execution_time <= 30
330
331 VARIABLES (execution_time, processEncoding.EncodingAdapter)
    unhandledRequest, (execution_time, processEncoding.execution_time)
    inState
332 VARIABLES (execution_time, processEncoding.EncodingAdapter)start, (
    execution_time, processEncoding.execution_time)end, (execution_time,
     processEncoding.EncodingAdapter)inCall
333
334 (execution_time, processEncoding.EncodingAdapter) Spec == INSTANCE
    execution_time
335 (execution_time, processEncoding.EncodingAdapter) PEOp2EtMapping.
    Mapping1 == [][execution_time, processEncoding]_MM
336
337
338 Spec == /\ processEncoding
339         /\ processEncoding, execution_time <= 30
340         /\ (execution_time, processEncoding.EncodingAdapter)
    PEOp2EtMapping.Mapping1
341         /\ (execution_time, processEncoding.EncodingAdapter)Spec!Spec
342         /\ [][]execution_time <=30]_p
```

```
343
344 ====
345 ---- MODULE MySqlClientApp ----
346 EXTENDS MySqlClientInterface, Naturals
347
348 VARIABLE internalMySqlClient
349 VARIABLE doHandle
350
351 InitauthenticateUser == /\ internalMySqlClient = 0
352                         /\ doHandle = 0
353                         /\ MySqlClientState \in InitialMySqlClientStates
354
355 ReceiveauthenticateUser == /\ authenticateUser(MySqlClientState,
    MySqlClientState')
356                            /\ doHandle = 0
357                            /\ doHandle'= 1
358                            /\ UNCHANGED internalMySqlClient
359
360 HandleauthenticateUser == /\ doHandle = 1
361                           /\ doHandle'= 2
362                           /\ UNCHANGED <<internalMySqlClient,
    MySqlClientState>>
363
364 ReplyStepauthenticateUser == /\ doHandle = 2
365                              /\ doHandle'= 0
366                              /\ SendData <<internalMySqlClient,
    MySqlClientState, MySqlClientState'>>
367                              /\ UNCHANGED internalMySqlClient
368
369 NextauthenticateUser == \/ ReceiveauthenticateUser
370                         \/ HandleauthenticateUser
371                         \/ ReplyStepauthenticateUser
372
373 vars == <<MySqlClientState, internalMySqlClient, doHandle>>
374
375 Spec == /\ InitauthenticateUser
376         /\ [NextauthenticateUser]_vars
377
378 authenticateUserSpec == /\ authenticateUser
379                         /\ (authenticateUser.MySqlClient)Spec!Spec
380
381 ----
382 InitgetUserAudioSubscriptions == /\ internalMySqlClient = 0
383                                  /\ doHandle = 0
384                                  /\ MySqlClientState \in
    InitialMySqlClientStates
385
386 ReceivegetUserAudioSubscriptions == /\ getUserAudioSubscriptions(
    MySqlClientState, MySqlClientState')
```

```
387                                     /\ doHandle = 0
388                                     /\ doHandle'= 1
389                                     /\ UNCHANGED internalMySqlClient
390
391 HandlegetUserAudioSubscriptions == /\ doHandle = 1
392                                     /\ doHandle'= 2
393                                     /\ UNCHANGED <<internalMySqlClient,
    MySqlClientState>>
394
395 ReplyStepgetUserAudioSubscriptions == /\ doHandle = 2
396                                     /\ doHandle'= 0
397                                     /\ SendData <<internalMySqlClient,
    MySqlClientState, MySqlClientState'>>
398                                     /\ UNCHANGED internalMySqlClient
399
400 NextgetUserAudioSubscriptions == \/ ReceivegetUserAudioSubscriptions
401                                  \/ HandlegetUserAudioSubscriptions
402                                  \/ ReplyStepgetUserAudioSubscriptions
403
404 getUserAudioSubscriptionsSpec == /\ InitgetUserAudioSubscriptions
405                                  /\ [NextgetUserAudioSubscriptions]_vars
406 ----
407 InitloadAudioFile == /\ internalMySqlClient = 0
408                      /\ doHandle = 0
409                      /\ MySqlClientState \in InitialMySqlClientStates
410
411 ReceiveloadAudioFile == /\ loadAudioFile(MySqlClientState,
    MySqlClientState')
412                         /\ doHandle = 0
413                         /\ doHandle'= 1
414                         /\ UNCHANGED internalMySqlClient
415
416 HandleloadAudioFile == /\ doHandle = 1
417                        /\ doHandle'= 2
418                        /\ UNCHANGED <<internalMySqlClient,
    MySqlClientState>>
419
420 ReplyStep loadAudioFile == /\ doHandle = 2
421                            /\ doHandle'= 0
422                            /\ SendData <<internalMySqlClient,
    MySqlClientState, MySqlClientState'>>
423                            /\ UNCHANGED internalMySqlClient
424
425 NextloadAudioFile == \/ ReceiveloadAudioFile
426                      \/ HandleloadAudioFile
427                      \/ ReplyStep loadAudioFile
428
429 loadAudioFileSpec == /\ InitloadAudioFile
430                      /\ [NextloadAudioFile]_vars
```

```
431 ----
432 InitstoreAudioFile == /\ internalMySqlClient = 0
433                     /\ doHandle = 0
434                     /\ MySqlClientState \in InitialMySqlClientStates
435
436 ReceivestoreAudioFile == /\ storeAudioFile(MySqlClientState,
    MySqlClientState')
437                          /\ doHandle = 0
438                          /\ doHandle'= 1
439                          /\ UNCHANGED internalMySqlClient
440
441 HandlestoreAudioFile == /\ doHandle = 1
442                         /\ doHandle'= 2
443                         /\ UNCHANGED <<internalMySqlClient,
    MySqlClientState>>
444
445 ReplyStepstoreAudioFile == /\ doHandle = 2
446             /\ doHandle'= 0
447             /\ SendData <<internalMySqlClient, MySqlClientState,
    MySqlClientState'>>
448             /\ UNCHANGED internalMySqlClient
449
450 NextstoreAudioFile == \/ ReceivestoreAudioFile
451                       \/ HandlestoreAudioFile
452                       \/ ReplyStepstoreAudioFile
453
454 Spec == /\ InitstoreAudioFile
455         /\ [NextstoreAudioFile]_vars
456
457 storeAudioFile, []execution_time <= 30
458
459 VARIABLES (execution_time, storeAudioFile.MySqlClient)unhandledRequest,
    (execution_time, storeAudioFile.execution_time)inState
460 VARIABLES (execution_time, storeAudioFile.MySqlClient)start, (
    execution_time, storeAudioFile.execution_time)end, (execution_time,
    storeAudioFile.MySqlClient)inCall
461
462 (execution_time, storeAudioFile.MySqlClient) Spec == INSTANCE
    execution_time
463 (execution_time, storeAudioFile.MySqlClient) SAOp2EtMapping.Mapping1 ==
    [][execution_time, storeAudioFile]_MM
464
465
466 storeAudioFileSpec == /\ storeAudioFile
467                       /\ storeAudioFile, execution_time <= 30
468                       /\ (execution_time, storeAudioFile.MySqlClient)
    SAOp2EtMapping.Mapping1
469                       /\ (execution_time, storeAudioFile.MySqlClient)
    Spec!Spec
```

```
470                         /\ [][]execution_time <=30]_p
471
472 Spec == /\ authenticateUserSpec
473        /\ getUserAudioSubscriptionsSpec
474        /\ loadAudioFileSpec
475        /\ storeAudioFileSpec
476
477 ====
```

Listing A.6: The TLA+ Specification for Components of Application.

A.2.3 TLA+ Specification of DB Scheduler

Listing A.7 shows the TLA+ specification of the *DB scheduler* context model.A DB Scheduler allocates the resource Database to various queries. *QueryCount* represents the number of queries that want to share the Database resource. A variable *AssignedTo* represents the number of the queries currently assigned the resource. Lines 10 thru 21 show the behaviour of DBScheduler and the actions like *Init*, *StartRequest* and *FinishRequest*.

```
1 ---------- MODULE DBScheduler ----------
2 EXTENDS Naturals
3
4 CONSTANT QueryCount
5 ASSUME (QueryCount \in Nat) /\ (QueryCount > 0)
6
7 VARIABLE AssignedTo
8 AssignedTo == {1..QueryCount}
9 ---------------------------
10 Init == AssignedTo \in AssignedToType
11
12 StartRequest == /\ inState = Idle
13                 /\ unhandledRequest = TRUE
14                 /\ inState' = HandlingRequest
15                 /\ unhandledRequest' = FALSE
16
17 FinishRequest == /\ inState = HandlingRequest
18                  /\ inState' = Idle
19                  /\ UNCHANGED unhandledRequest
20
21 Next == StartRequest
22
23 DBScheduler == /\ Init
```

```
24                    /\ [] [StartRequest /\ FinishRequest /\ Next]_AssignedTo
25 ============================
```

Listing A.7: The TLA+ specification for *DB scheduler*

A.2.4 TLA+ Specification of Web Audio Container

Listing A.8 shows the TLA+ specification of *Web Audio Store Container* module. It is equivalent to QML/CS specification of the *Web Audio Store Application* discussed in listing 8.5. Lines 1 thru 74 show the Web Audio Store container specification and manage a number of component instances in order to reach a response time time with them. Line 2 shows that this module extends *Real Time* module. Lines 4 thru 8 show that a container has two parameters *ResponseTime* and *ExecutionTime*. *ResponseTime* refers to the response time the container aims to achieve whereas *ExecutionTime* refers to the execution time of the components available. In addition, it includes variables like *QueryCount* and *met*. *QueryCount* represents the number of queries and *met* represents maximum execution time allowed.

```
1  ---------- MODULE AudioSystemContainer ----------
2  EXTENDS RealTime
3
4  CONSTANT ResponseTime
5  ASSUME (ResponseTime \in Real) /\ (ResponseTime > 0)
6
7  CONSTANT ExecutionTime
8  ASSUME (ExecutionTime \in Real) /\ (ExecutionTime > 0)
9
10 ----------------------------
11 VARIABLES QueryCount, met
12 ----------------------------
13 VARIABLES DBMinExecTime, DBAssignedTo
14 _QueryScheduler (QueryCountConstraints, metConstraints) == INSTANCE
       TimedDBScheduler
15                 WITH    MinExecTime <- DBMinExecTime,
16                         AssignedTo <- DBAssignedTo,
17                         QueryCount <- QueryCountConstraints
18                         met<- metConstraints
19 DBCanSchedule (QueryCountConstraints, metConstraints) ==
20                                     /\_QueryScheduler (QueryCount, met)
21                                     !TimedDBScheduler
22                                     /\ []_QueryScheduler (QueryCount, met)
23                                     !ExecutionTimesOk
24 ----------------------------
25 VARIABLES RTLastResponseTime, RTinState, RTUnhandeledRequest
```

```
26 _RTResponseTime(ResponseTimeConstraint) == INSTANCE
       ResponseTimeConstraintedRT
27                                                 WITH
28                                                 ResponseTime <-
       ResponseTimeConstraint,
29                                                 LastResponseTime <-
       RTLastResponseTime,
30                                                 UnhandeledRequest<-
       RTUnhandeledRequest
31 RTResponseTime (ResponseTimeConstraint) == _RTResponseTime (
       ResponseTimeConstraint) !RT
32 ----------------------------
33 AudioSystemContainerPreCond ==
34                  /\ ExecutionTime <= ResponseTime
35                  /\ DBCanSchedule (1, [n \in |->ResponseTime],
36                      [n \in |->ExecutionTime])
37                 /\ ComponentMaxExecTime(ExecutionTime)
38                 /\ MinInterrequestTime (ResponseTime)
39 ----------------------------
40 AudioSystemContainerPostCond ==
41                  /\ RTResponseTime (ResponseTime)
42                  /\ QueryCount = 1
43                  /\ met = [n \in |->ExecutionTime]
44 ----------------------------
45 AudioSystemContainer ==
46              AudioSystemContainerPreCond -+->
       AudioSystemContainerPostCond
47 ============================
```

Listing A.8: The TLA+ specification for *Web Audio Store Container specification*

A.2.5 TLA+ Specification of Web Audio System

Listing A.9 shows the TLA+ specification of *Web Audio Store System* module. It is equivalent to QML/CS specification of the *Web Audio Store Application* discussed in listing 8.6. Lines 1 thru 74 show the Web Audio Store system specification, which contains a number of components with an execution time of 30 milliseconds, a MySql scheduled DB, and an Audio System container. Line 3 extends *Real Time* module and line 8 defines a variable *now*, which refers to the current time. In addition, it includes the variables for resource MyDB, system's container. Lines 58 thru 67 demonstrates the service of the system that is to perform. Lines 71 thru 83 shows the complete system specification.

```
1 ---- MODULE WebAudioStoreApplicationSpecification ----
2 EXTENDS Real
```

```
3
4  CONSTANT ResponseTime
5  ASSUME (ResponseTime \in Real) /\ (ResponseTime > 0)
6
7  VARIABLE now
8  ----
9  VARIABLES ComponentAudioStoreLastExec, ComponentAudioStoreInState,
       ComponentAudioStoreUnhandledRequest
10 _ComponentAudioStore (ExecutionTime) == INSTANCE
       ExecTimeConstrainedComponent WITH
11                     LastExecutionTime <- ComponentAudioStoreLastExec,
12                     inState <- ComponentAudioStoreInState,
13                     unhandledRequest <-
       ComponentAudioStoreUnhandledRequest
14 ComponentAudioStore (ExecutionTime) == _ComponentAudioStore(
       ExecutionTime)!Component
15 ----
16 VARIABLES ComponentWebFormLastExec, ComponentWebFormInState,
       ComponentWebFormUnhandledRequest
17 _ComponentWebForm (ExecutionTime) == INSTANCE
       ExecTimeConstrainedComponent WITH
18                     LastExecutionTime <- ComponentWebFormLastExec,
19                     inState <- ComponentWebFormInState,
20                     unhandledRequest <- ComponentWebFormUnhandledRequest
21 ComponentWebForm (ExecutionTime) == _ComponentWebForm(ExecutionTime)!
       Component
22 ----
23 VARIABLES ComponentMySqlClientLastExec, ComponentMySqlClientInState,
       ComponentMySqlClientUnhandledRequest
24 _ComponentMySqlClient (ExecutionTime) == INSTANCE
       ExecTimeConstrainedComponent WITH
25                       LastExecutionTime <- ComponentMySqlClientLastExec,
26                       inState <- ComponentMySqlClientInState,
27                       unhandledRequest <-
       ComponentMySqlClientUnhandledRequest
28 ComponentMySqlClient (ExecutionTime) == _ComponentMySqlClient (
       ExecutionTime)!Component
29 ----
30 VARIABLES MYDB_MinExecTime, MYDB_AssignedTo
31
32 _MyDB (QueryCount, met) == INSTANCE SQLScheduler WITH
33                            MinExecTime <- MYDB_MinExecTime,
34                            AssignedTo <- MYDB_AssignedTo
35 MyDB (QueryCount, met) ==
36                          _MyDB (QueryCount, met)!SQLScheduler
37 ----
38 VARIABLES SCDBMinExecTime, SCDBAssignedTo
39 VARIABLES SCCmpInState, SCCmpUnhandledRequest, SCCmpLastExecutionTime
40 VARIABLES SCServLastResponseTime, SCServInState, SCServUnhandledRequest
```

```
41 _AudioSystemContainer (ExecutionTimeConstr, ResponseTimeConstr,
42                 QueryCount, met)
43     == INSTANCE AudioSystemContainer
44        WITH ExecutionTime <- ExecutionTimeConstr,
45        ResponseTime <- ResponseTimeConstr,
46        DBMinExecTime <- SCDBMinExecTime,
47        DBAssignedTo <- SCDBAssignedTo,
48        CmpInState <- SCCmpInState,
49        CmpUnhandledRequest <- SCCmpUnhandledRequest,
50        CmpLastExecutionTime <- SCCmpLastExecutionTime,
51 AudioSystemContainer (ExecutionTimeConstr, ResponseTimeConstr,
    QueryCount,met)
52     == _AudioSystemContainer(ExecutionTimeConstr, ResponseTimeConstr,
53                       QueryCount, met)!AudioSystemContainer
54 ----
55 _SystemService (ResponseTimeConstraint)
56     == INSTANCE ResponseTimeConstrainedService
57        WITH ResponseTime <- ResponseTimeConstraint,
58             LastResponseTime <- ServLastResponseTime,
59             inState <- ServInState,
60             unhandledRequest <- ServUnhandledRequest
61 SystemService (ResponseTimeConstraint) == _SystemService(
    ResponseTimeConstraint)!Service
62 ----
63 VARIABLES DBQueryCount, DBmet
64 VARIABLES SCQueryCount, SCmet
65 System == /\ ComponentAudioStore (20)
66           /\ ComponentMySqlClient (30)
67           /\ ComponentWebForm(20)
68           /\ MyDB (DBQueryCount, DBmet)
69           /\ AudioSystemContainer (30, ResponseTime, SCQueryCount, SCmet
    )
70           /\ [] /\ ServLastResponseTime = SCServLastResponseTime
71                 /\ ServInState = SCServInState
72                 /\ ServUnhandledRequest = SCServUnhandledRequest
73           /\ [] /\ MYDB_MinExecTime = SCDBMinExecTime
74                 /\ MYDB_AssignedTo = SCDBAssignedTo
75                 /\ DBQueryCountCount = SCQueryCount
76                 /\ DBmet = SCmet
77 ====
```

Listing A.9: The TLA+ specification for *Web Audio Store System specification*

A.3 Translational Semantics Template of TLA+ specification

A.3.1 Context Model Template

Defining the semantics of QML/CS: Definition of a context model on which to base our semantics definition. As we already stated in Section 3.3.1, specifications of non-functional properties can only be defined relative to a context model. Context Model translation template.

```
1 [%var states :List;%][%var transName :List;%] [%var transAtt :List;%] [%
    var transRef :List;%] [% transName.clear();%]
2 [%for ( sm in ContextModel!StateMachineModel) {%]
3 [%for (st in sm.states) {%] [%states.add(st.name);%][%}%][%for (tr in sm
    .transitions) {%] [%transName.add(tr);%]
4 [%for (attrName in tr.eClass.EAttributes){%]
5 [%transAtt.add(attrName.name);%][%}%]
6 [%for (attrRef in tr.eClass.EReferences){%][%transRef.add(attrRef.name)
    ;%][%}%][%}%]
7 ---------- MODULE [%=sm.name%] ----------
8
9 VARIABLE [%=transRef[3]%]
10 VARIABLE [%=transRef[4]%]
11 VARIABLE [%=transAtt[1]%]
12 VARIABLE [%=transAtt[2]%]
13 vars == <<[%=transRef[3]%], [%=transRef[4]%], [%=transAtt[1]%], [%=
    transAtt[2]%]>>
14
15 ----------
16 InitEnv == [%=transAtt[1]%] == FALSE
17
18 RequestArrival == /\ [%=transAtt[1]%] == FALSE
19             /\ [%=transAtt[2]%] == TRUE
20             /\ UNCHANGED [%=transRef[3]%]
21
22 [%=sm.name%]Agent == \/ /\ [%=transAtt[1]%] == TRUE
23                 /\ [%=transAtt[1]%]'== FALSE
24                 \/ not UNCHANGED [%=transRef[3]%]
25
26 EnvSpec == /\ InitEnv
27         /\ [] [RequestArrival \/ [%=sm.name%]Agent]_vars
28 ----------
29
30
31 [% for (tt in transName){%][% if (tt == transName[0] or tt == transName.
    get(transName.size()-1)){%]
```

```
[%=tt.name%] == [%=transRef[3]%] =[% if (tt.instate <> null) {%] "[%=
    tt.instate.name%]"[%}%]
        /\
[%}%]

[%}%]

[*
[%=transName[0].name%] == [%=transRef[3]%] = "[%=transName[0].instate.
    name%]"

[%=transName[2].name%] == /\ [%=transRef[3]%] = "[%=states[0]%]"
        /\ [%=transAtt[1]%] = [%=transName[2].unhandledRequest%]
                /\ inState' = "[%=states[2]%]"
                /\ [%=transAtt[1]%]' = [%=transName[2].unhandledRequest
    %]

[%=transName[3].name%] == /\ [%=transRef[3]%] = "[%=states[2]%]"
                  /\ [%=transRef[3]%]' = "[%=states[0]%]"
                  /\ UNCHANGED [%=transAtt[1]%]


Next == [%=transName[2].name%] \/ [%=transName[3].name%]
*]

Spec == /\ [%=transName[0].name%]
        /\ [] [Next /\ EnvAgent]_vars
        [% transName.clear();%]
=======
[%}%]

[*
[%for (ms in qmlcs!MeasurementDeclaration.all) {%]


------------- MODULE [%=ms.name%] -------------
[%}%]
*]
```

Listing A.10: Context Model EGL Template for the Semantic Translation.

A.3.2 Measurement Template

Defining the semantics of QML/CS: Measurement translation template.

```
[%for (ms in qmlcs!MeasurementDeclaration.all) {%]
....
------------- MODULE [%=ms.name%] -------------
EXTENDS RealTime
```

```
[%for ( mcxt in ms.msCxt) {%]
[%for ( mcxt2 in mcxt) {%]
  [%for ( mcxt3 in mcxt2.variable) {%]
    [%for ( mcxt4 in mcxt3.ownedConstraint.specification.ownedExpression
    .ownedExpression){%]
      [%for ( mcxt5 in mcxt4) {%]
      [%if(mcxt5.eClass.name="AttributeExpCS"){%]
          [%for (mcxt6 in mcxt5.name){%]
          [% if (variableSet.add(mcxt6.name)) variables.add(mcxt6.name)
    ;%]
          [%}%]
      [%}%]
      [%}%]
    [%}%]
  [%}%]
[%}%]
[%}%]

[%for (fp in ms.params){%]
  [%  if (fp.type.name="ServiceOperation"){%]
[%=fp.name%] == INSTANCE Service
  [%}%]
  [% if (fp.type.name="ComponentOperation"){%]
[%=fp.name%] == INSTANCE Component
  [%}%]
[%}%]
-------------------------------------------------
VARIABLES [% for (myvar in variables){ %] [%=myvar%] [%if (myvar <>
    variables.get(variables.size()-1)){%],[%}}%]
VARIABLES unhandledRequest, inState
-------------------------------------------------

[%for (mcxt in ms.msCxt) {%]
      [%for (mcxtTrans in mcxt.Trans) {%]
[% if (mcxt <> (ms.msCxt[0])){%]On[%}%][%=mcxtTrans.name%][%for (
    mcxtpara in mcxt.mpara) {%] == [% if (mcxt <> (ms.msCxt[0])){%][%=
    mcxtpara.name%][%}}%][% if (mcxt <> (ms.msCxt[0])){%]![%=mcxtTrans.
    name%] =>[%} else {%]/\ [%=variables.get(0)%] \in [%=ms.dtype%]
    [%}%]
 [%for ( mcxtpara2 in mcxt) {%]
  [%for ( vari in mcxtpara2.variable) {%] [%for ( mcxtExpr222 in vari.
    ownedConstraint.specification) {%][%}%][%for ( mcxtExpr2 in vari.
    ownedConstraint.specification.ownedExpression) {%]
      [%var operatorShown : Integer =0;%]
      [%var varUsed : Integer =0;%]
        [%for ( mcxtExpr3 in mcxtExpr2.ownedExpression){%][% if(
    mcxtExpr3.eClass.name ="AttributeExpCS") {%][%for (mcxtExprName in
    mcxtExpr3.name){%] [% if (mcxt == (ms.msCxt[0])){%][%if (
    operatorShown==0){%][%}%] [%}%][%if (operatorShown==0){%]
```

```
42          /\[%}%] [%=mcxtExprName.name%][% for (dd in displayedList) if (
    dd == mcxtExprName.name ) varUsed=1; %] [% if (varUsed==0)
    displayedList.add(mcxtExprName.name); %][% if (varUsed==1 and
    operatorShown==0){ %]'[% } %][%}%][%if (operatorShown==0){%] [%for (
     mcxtExprRef in mcxtExpr2.eClass.EReferences) {%][% if (mcxtExprRef.
    name == "ownedOperator") {%] [%for (OperName in mcxtExpr2.
    ownedOperator.name){%][%=OperName%][%operatorShown
    =1;%][%}%][%}%][%}%][%}%][%}%][%if( mcxtExpr3.eClass.name="
    VariableValue") {%][%=mcxtExpr3.name%][%} else if ( mcxtExpr3.eClass
    .name="NumberLiteralExpCS") {%][%=mcxtExpr3.name%]  [%} else if (
    mcxtExpr3.eClass.name="BooleanLiteralExpCS") {%][%=mcxtExpr3.name
    %][%}%]
43          [%}%]
44      [%}%]
45      [%}%]
46      [%}%]
47      [%}%][%}%]
48 [% variables.clear(); %]
49 [%for (fp in ms.params){%]
50
51
52 Spec == /\  [%  if (fp.type.name="ServiceOperation"){%] [%=fp.name%]!
    Service [%}%]  [% if (fp.type.name="ComponentOperation"){%][%=fp.
    name%]! Component[%}%]
53 [%for ( mcxt in ms.msCxt) {%]
54 [%for ( mcxtTrans in mcxt.Trans) {%]
55
56   [% if (mcxt == (ms.msCxt[ms.msCxt.size()-1])){%]
57          /\ [][On[%=mcxtTrans.name%]]_([%=ms.name%])    [%}
58   else{%]
59          /\ On[%=mcxtTrans.name%][%}%]
60     [%}%][%}%][%}%]
61
62 ============================
63 [%}%]
```

Listing A.11: Measurement EGL Template for the Semantic Translation of Service Operations and Component Operations.

A.4 QML/CS Prototype Screenshots and Code sample

The figures A.16 and A.17 show the editor environment for the tool developed in this thesis. The images show different components of the tool a user can interact with to write the specification. It indicates hierarchy of the concepts as they exist in the models and the QML/CS code along with showing mapping and association of

the concepts in QML/CS to the relevant models. Also, the figures show the model and the layout of how different states are handled in the models and the user can change the sequence of the states as well as their mapping to confirm to the model and requirements of the application being used to specify.

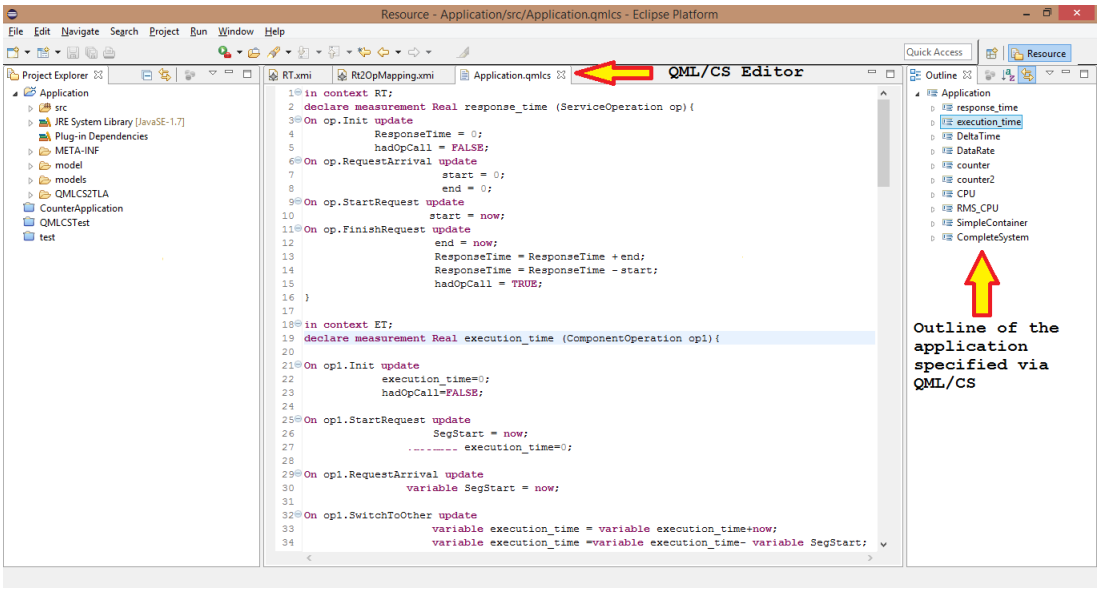

Figure A.16: QML/CS Prototype Screenshot 1

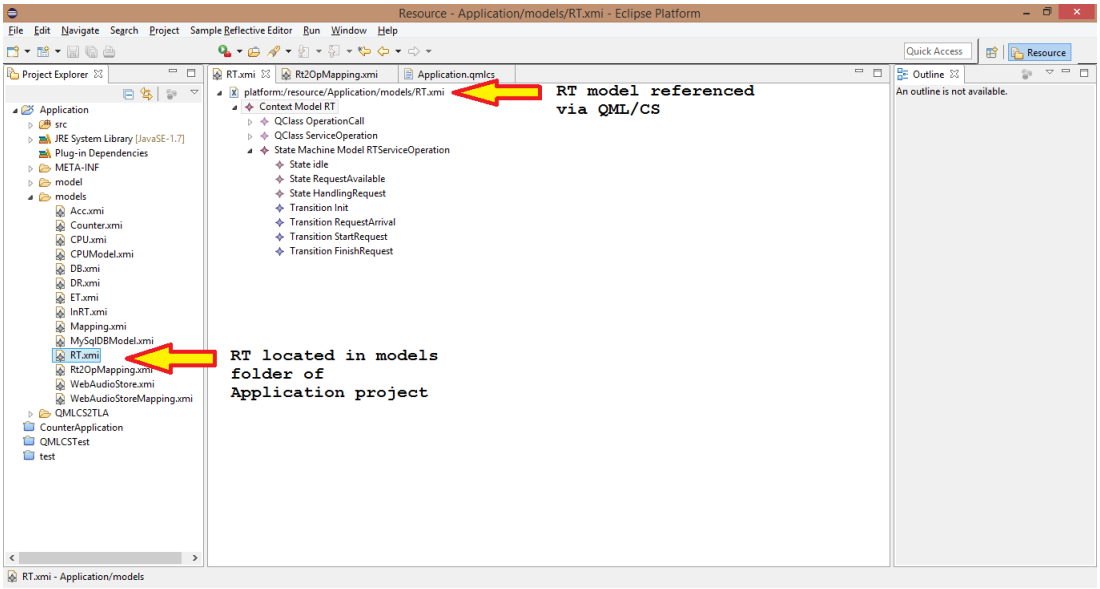

Figure A.17: QML/CS Prototype Screenshot 2

B
Appendix

B.1 Complete Grammar of QML/CS language in Xtext

In this appendix we have gathered the full QML/CS language grammar.

B.1.1 Grammar for QML/CS

QML/CS grammar consists of a number of declaration a number of declaration as follows:

1. Measurement Declaration:

   - Syntax: Example *ResponseTime* as can be seen in B.1:

```
in context RT;
declare measurement real response_time (ServiceOperation op){
  spec op.invocations->last.end-op.invocations->last.start;
}
```

Listing B.1: Measurement Definition Syntax in qmlcs language

   - Measurement Declaration Grammar as can be seen in B.2:

```
MeasurementDeclaration :
.....
  'declare' 'measurement' Type name = MeasurementID '('
.....
  (ownedConstraint+=specConstraints)*
'}';
```

Listing B.2: Measurement Grammar in Xtext

2. ComponentService Declaration:

   - Syntax: Example *Counter* Application as can be seen in B.3:

```
application CounterModel;
declare Service counter {
        provides Operation int getData();

       always response_time((getData by GD2RTMapping.Mapping1))
     <60;
}
```

Listing B.3: Application Definition Syntax in qmlcs language

   - Component Service Declaration Grammar as can be seen in B.4:

```
ApplicationModelStatement:
  'application' name=ImportName ';';


ComponentServiceDeclaration:
  model = ApplicationModelStatement
  'declare' comOrServ=('Component'|'Service') name =
    ComponentOrServiceNameID '{'
  (('uses'|'provides') (op += ApplicationOperation) '()' ';')+
  (ownedConstraint+=OCLConstraint)*'}';

ApplicationOperation:
{ApplicationOperation}
  type=[qmlcsmm::QClass|UnrestrictedName] returnType=Type
  name=ApplicationOperationId;

ApplicationOperationId returns ecore::EString:
  ID;
```

Listing B.4: Service Grammar in Xtext

B.1.2 QML/CS grammar

```
grammar org.xtext.example.qmlcs.QMLCS
hidden(WS, ML_COMMENT, SL_COMMENT)

import "http://www.eclipse.org/emf/2002/Ecore" as ecore
import "http://www.abdu.org/qmlcsmm" as qmlcsmm
generate qMLCS "http://www.xtext.org/example/qmlcs/QMLCS"

/* */
QMLCSModel :
  (DSLelements += QMLCSElements)*

;

QMLCSElements :
  GlobalStatement | DeclarationStatement
;


GlobalStatement :
  ApplicationModelStatement // | Class
;


Type:
  'boolean' | 'int' | 'real' | 'string'
;


DeclarationStatement:
    MeasurementDeclaration | ComponentServiceDeclaration|
    AbstractResourceDeclaration | ConcreteResourceDeclaration|
    ContainerDeclaration|SystemDeclaration
;


ApplicationModelStatement :
  'application' name=ImportName ';'
;

InContextModelStatement:
  'in' 'context' name=ImportName';'
;

ImportName returns ecore::EString:
  ID ('.' ID)* '.*'?
;


MeasurementExp:
```

```
50     (measurement = [MeasurementDeclaration]) ( '()' | '('
51     ((argument+=MeasurementExpArgument) (',' (argument+=
       MeasurementExpArgument))*)?
52    ')')
53 ;
54
55 MeasurementExpArgument:
56    name = [ApplicationOperation] 'by' mapping=ID
57 ;
58
59 MeasurementDeclaration :
60       context = InContextModelStatement
61       'declare' 'measurement' Type name = MeasurementID '('
62       (params+= MeasurementParam
63      (',' params+=MeasurementParam)*
64      )?
65    ')' '{'
66           (ownedConstraint+=specConstraints)*
67      '}'
68    ;
69
70 MeasurementParam:
71    {MeasurementParam} (type=[qmlcsmm::QClass|UnrestrictedName]
72           name=MeasurementArgumentId)
73 ;
74
75
76
77 MeasurementArgumentId returns ecore::EString:
78    ID
79 ;
80
81
82 MeasurementID:
83    name=ID
84 ;
85
86
87
88 ComponentServiceDeclaration:
89    model = ApplicationModelStatement
90    'declare' decl=('Component'|'Service') CponentOrServiceName = ID '{'
91    ('provides' (op += ApplicationOperation) '()' ';')+
92     (ownedConstraint+=alwaysConstraint)*
93    '}'
94 ;
95
96 ApplicationOperation:
97    {ApplicationOperation}
```

```
98    type=[qmlcsmm::QClass|UnrestrictedName] returnType=Type
99    name=ApplicationOperationId
100 ;
101
102 ApplicationOperationId returns ecore::EString:
103   ID
104 ;
105
106 AbstractResourceDeclaration:
107   context = InContextModelStatement
108
109   'declare' 'abstract' decl='resource' name=AbstractresourceNameID '{'
110   'demand' type=DemandType ';'
111    (service +=AbstractResourceService)+
112    (ownedConstraint+=alwaysConstraint)+
113 '}'
114 ;
115
116 DemandType:
117   type=[qmlcsmm::QClass|UnrestrictedName]
118 ;
119
120 AbstractresourceNameID returns ecore::EString:
121   ID
122 ;
123 AbstractResourceService:
124   'service' '(' ownedType = CollectionTypeCS name=ID ')' '='
125 ;
126
127 ConcreteResourceCapacity:
128   'capacityLimit' '(' ownedType = CollectionTypeCS name=ID ')' '='
129 ;
130
131 ConcreteResourceDeclaration:
132   'declare' 'resource' ConResourceName = ID 'of' abstractRes=[
      AbstractResourceDeclaration|AbstractresourceNameID] '{'
133    (capacities+=ConcreteResourceCapacity)+
134     (ownedConstraint+=Constraints)+
135    '}'
136 ;
137
138
139 ContainerDeclaration:
140   'declare' decl='container' name=ContainerNameID '('(params+=
      ContainerParam (',' params+=ContainerParam)*)? ')' '{'
141    (HelperVariable+=HelperVariables (','HelperVariable+=HelperVariables
      )*)?
142
```

```
143     (('requires' 'component') (componentre += componentrequried(',' componentrequre += componentrequried)*))*
144      'resource' abstractRes=[AbstractResourceDeclaration|AbstractresourceNameID] '.' ResFunction=[qmlcsmm::Function|UnrestrictedName] '('
145         '('
146         ((ownedConstraint+=SpecificationCS) (',' ownedConstraint+=SpecificationCS)* )?
147          ')'
148           ')'';'
149           'provides' 'service' 'implemented by' serv=[ServiceDeclaration|ComponentOrServiceNameID]'{'
150                  (ownedConstraint+=(Constraints|alwaysConstraint))*
151
152    '}'
153
154 '}'
155 ;
156
157 ContainerName:
158      name=ID ;
159
160 HelperVariables:
161   ID':'Type ';' ;
162
163 ContainerParam:
164   ID':'Type;
165
166 S
167 SystemDeclaration:
168    'System' name=ID '{'
169       ('instance' (instanceList += InstanceList) ID ';')*
170
171        ('container'
172       'uses' (ComponentsAndResourcesUsed += ComponentsAndResources(',' ComponentsAndResourcesUsed += ComponentsAndResources)*)';')*
173
174
175       'container'
176       'provides'
177        (serviceProvided+=ServiceProvided';')*
178      '}'
179 ;
180 ContainerParamValue:
181 name=[ContainerDeclaration|ContainerNameID] '('val=NUMBER_LITERAL')'
182 ;
183
184 InstanceList:
```

```
185 name=[ServiceDeclaration|ComponentOrServiceNameID]| name=[
      AbstractResourceDeclaration|AbstractresourceNameID]|
      ContainerParamValue
186 ;
187
188 ComponentsAndResources:
189  name=([ServiceDeclaration|ComponentOrServiceNameID] |[
      AbstractResourceDeclaration|AbstractresourceNameID])
190 ;
191 ServiceProvided:
192    services=ID //services=[ecore::EClass|UnrestrictedName]
193 ;
```

Listing B.5: QML/CS grammar in Xtext

B.1.3 Extended OCL grammar

```
1  Model returns ContextCS:
2    ownedExpression=ExpCS;
3
4  terminal fragment ESCAPED_CHARACTER:
5    '\\' ('b' | 't' | 'n' | 'f' | 'r' | 'u' | '"' | "'" | '\\');
6
7  terminal fragment LETTER_CHARACTER:
8    'a'..'z' | 'A'..'Z' | '_';
9
10 terminal STRING :
11       '"' ( '\\' . /* 'b'|'t'|'n'|'f'|'r'|'u'|'"'|"'"|'\\' */ | !('\\'|'
       "') )* '"' |
12       "'" ( '\\' . /* 'b'|'t'|'n'|'f'|'r'|'u'|'"'|"'"|'\\' */ | !('\\'|"
       '") )* "'";
13
14 terminal ID      : '^'?('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'_'|'0'
       ..'9')*;
15 terminal INT: // String to allow diverse re-use
16   ('0'..'9')+;    // multiple leading zeroes occur as floating point
       fractional part
17
18 LOWER returns ecore::EInt:
19   INT
20 ;
21
22 UPPER returns ecore::EInt:
23   INT | '*'
24 ;
25
26 NUMBER_LITERAL returns BigNumber: // Not terminal to allow parser
       backtracking to sort out "5..7"
```

```
27    {BigNumber}INT; // EssentialOCLTokenSource pieces this together ('.'
       INT)? (('e' | 'E') ('+' | '-')? INT)?;
28
29  terminal ML_COMMENT:
30    '/*' ->'*/';
31
32  terminal SL_COMMENT:
33    '--' !('\n' | '\r')* ('\r'? '\n')?;
34
35  terminal WS:
36    (' ' | '\t' | '\r' | '\n')+;
37
38  terminal ANY_OTHER:
39    .;
40
41  URI:
42    //SINGLE_QUOTED_STRING;
43    STRING;//DOUBLE_QUOTED_STRING;
44
45  EssentialOCLReservedKeyword:
46    'and'
47    | 'else'
48    | 'endif'
49    | 'if'
50    | 'implies'
51    | 'in'
52    | 'let'
53    | 'not'
54    | 'or'
55    | 'then'
56    | 'xor';
57
58  EssentialOCLUnaryOperatorCS returns UnaryOperatorCS:
59    name=('-' | 'not');
60
61  EssentialOCLInfixOperatorCS returns BinaryOperatorCS:
62    name=('*' | '/' | '+' | '-' | '>' | '<' | '>=' | '<=' | '=' | '<>' | '
       and' | 'or' | 'xor' | 'implies');
63
64  EssentialOCLNavigationOperatorCS returns NavigationOperatorCS:
65    name=('.' | '->');
66
67  Identifier:
68    ID;
69
70  StringLiteral:
71    STRING;//SINGLE_QUOTED_STRING;
72
73  BinaryOperatorCS returns BinaryOperatorCS:
```

```
74    InfixOperatorCS | NavigationOperatorCS;
75
76  InfixOperatorCS returns BinaryOperatorCS:        // Intended to be
      overrideable
77    EssentialOCLInfixOperatorCS;
78
79  NavigationOperatorCS returns NavigationOperatorCS:   // Intended to be
      overrideable
80    EssentialOCLNavigationOperatorCS;
81
82  UnaryOperatorCS returns UnaryOperatorCS:        // Intended to be
      overrideable
83    EssentialOCLUnaryOperatorCS;
84
85  EssentialOCLUnrestrictedName returns ecore::EString:
86    Identifier;
87
88  UnrestrictedName returns ecore::EString: // Intended to be overridden
89    EssentialOCLUnrestrictedName;
90
91  EssentialOCLUnreservedName returns ecore::EString:
92    UnrestrictedName
93  | CollectionTypeIdentifier
94  | PrimitiveTypeIdentifier
95  | 'Tuple'
96  ;
97
98  UnreservedName returns ecore::EString: // Intended to be overridden
99    EssentialOCLUnreservedName;
100
101 PathNameCS returns PathNameCS:
102   path+=FirstPathElementCS ( '::' path+=NextPathElementCS)*;
103
104 FirstPathElementCS returns PathElementCS:
105   element=[ecore::ENamedElement|UnrestrictedName] |
106   element=[MeasurementParam|MeasurementArgumentId] |
107   element=[qmlcsmm::AssociationEndList|UnrestrictedName] |
108   element=[qmlcsmm::OCLOperation|UnrestrictedName];
109
110 NextPathElementCS returns PathElementCS:
111   element=[ecore::ENamedElement|UnrestrictedName];
112
113 URIPathNameCS returns PathNameCS:
114   path+=URIFirstPathElementCS ('::' path+=NextPathElementCS)*;
115
116 URIFirstPathElementCS returns PathElementCS:
117   element=[ecore::ENamedElement|UnrestrictedName] |  element=[ecore::
      ENamedElement|URI];
118
```

```
119
120 PrimitiveTypeIdentifier:
121   'Boolean'
122   | 'Integer'
123   | 'Real'
124   | 'String'
125   | 'UnlimitedNatural'
126   | 'OclAny'
127   | 'OclInvalid'
128   | 'OclVoid';
129
130 PrimitiveTypeCS returns PrimitiveTypeRefCS:
131   name=PrimitiveTypeIdentifier;
132
133 CollectionTypeIdentifier returns ecore::EString:
134   'Set'
135   | 'Bag'
136   | 'Sequence'
137   | 'Collection'
138   | 'OrderedSet';
139
140 CollectionTypeCS returns CollectionTypeCS:
141   name=CollectionTypeIdentifier ('(' ownedType=TypeExpCS ')')?;
142
143 MultiplicityBoundsCS returns MultiplicityBoundsCS:
144   lowerBound=LOWER ('..' upperBound=UPPER)?;
145
146 MultiplicityCS returns MultiplicityCS:
147   '[' (MultiplicityBoundsCS | MultiplicityStringCS) ']';
148
149 MultiplicityStringCS returns MultiplicityStringCS:
150   stringBounds=('*'|'+'|'?');
151
152 TupleTypeCS returns TupleTypeCS:
153   name='Tuple' ('(' (ownedParts+=TuplePartCS (',' ownedParts+=
      TuplePartCS)*)? ')')?;
154
155 TuplePartCS returns TuplePartCS:
156   name=UnrestrictedName ':' ownedType=TypeExpCS;
157
158 CollectionLiteralExpCS returns CollectionLiteralExpCS:
159   ownedType=CollectionTypeCS
160   '{' (ownedParts+=CollectionLiteralPartCS
161   (',' ownedParts+=CollectionLiteralPartCS)*)?
162   '}';
163
164 CollectionLiteralPartCS returns CollectionLiteralPartCS:
165   expressionCS=ExpCS ('..' lastExpressionCS=ExpCS)?;
166
```

```
167 ConstructorPartCS returns ConstructorPartCS:
168   property=UnrestrictedName '=' initExpression=ExpCS;
169
170 PrimitiveLiteralExpCS returns PrimitiveLiteralExpCS:
171   NumberLiteralExpCS
172   | StringLiteralExpCS
173   | BooleanLiteralExpCS
174   | UnlimitedNaturalLiteralExpCS
175   | InvalidLiteralExpCS
176   | NullLiteralExpCS;
177
178 TupleLiteralExpCS returns TupleLiteralExpCS:
179   'Tuple' '{' ownedParts+=TupleLiteralPartCS (',' ownedParts+=
      TupleLiteralPartCS)* '}';
180
181 TupleLiteralPartCS returns TupleLiteralPartCS:
182   name=UnrestrictedName (':' ownedType=TypeExpCS)? '=' initExpression=
      ExpCS;
183
184 NumberLiteralExpCS returns NumberLiteralExpCS:
185   name=NUMBER_LITERAL;
186
187 StringLiteralExpCS returns StringLiteralExpCS:
188   name+=StringLiteral+;
189
190 BooleanLiteralExpCS returns BooleanLiteralExpCS:
191   name='true'
192   | name='false';
193
194 UnlimitedNaturalLiteralExpCS returns UnlimitedNaturalLiteralExpCS:
195   {UnlimitedNaturalLiteralExpCS} '*';
196
197 InvalidLiteralExpCS returns InvalidLiteralExpCS:
198   {InvalidLiteralExpCS} 'invalid';
199
200 NullLiteralExpCS returns NullLiteralExpCS:
201   {NullLiteralExpCS} 'null';
202
203 TypeLiteralCS returns TypedRefCS:
204   PrimitiveTypeCS
205   | CollectionTypeCS
206   | TupleTypeCS;
207
208 TypeLiteralWithMultiplicityCS returns TypedRefCS:
209   TypeLiteralCS multiplicity=MultiplicityCS?;
210
211 TypeLiteralExpCS returns TypeLiteralExpCS:
212   ownedType=TypeLiteralWithMultiplicityCS;
213
```

```
214 TypeNameExpCS returns TypeNameExpCS:
215   pathName=PathNameCS;
216
217 TypeExpCS returns TypedRefCS:
218   name=ID multiplicity=MultiplicityCS?;
219
220 ExpCS returns ExpCS:
221   (PrefixedExpCS
222   ({InfixExpCS.ownedExpression+=current} ownedOperator+=BinaryOperatorCS
223   ( (ownedExpression+=PrefixedExpCS
224   (ownedOperator+=BinaryOperatorCS ownedExpression+=PrefixedExpCS)*
225   (ownedOperator+=BinaryOperatorCS ownedExpression+=LetExpCS)?
226   )
227   | (ownedExpression+=LetExpCS)
228   )
229   )?
230   )
231 |   ({PrefixExpCS} ownedOperator+=UnaryOperatorCS+ ownedExpression=
      LetExpCS)
232 | LetExpCS;
233
234 PrefixedExpCS returns ExpCS:
235   ({PrefixExpCS} ownedOperator+=UnaryOperatorCS+ ownedExpression=
      PrimaryExpCS)
236 |   PrimaryExpCS;
237
238
239 PrimaryExpCS returns ExpCS:
240   NestedExpCS
241 |   MeasurementExp
242 | IfExpCS
243 |   SelfExpCS
244 |   PrimitiveLiteralExpCS
245 |   TupleLiteralExpCS
246 |   CollectionLiteralExpCS
247 |   TypeLiteralExpCS
248 |   ({NameExpCS} pathName=PathNameCS
249   (
250   (({IndexExpCS.nameExp=current} '[' firstIndexes+=ExpCS (',')
      firstIndexes+=ExpCS)* ']'
251   ('[' secondIndexes+=ExpCS (',' secondIndexes+=ExpCS)* ']')?
252   (atPre?='@' 'pre')?
253   ))
254   |({ConstructorExpCS.nameExp=current} '{'
255   ( ((ownedParts+=ConstructorPartCS (',' ownedParts+=ConstructorPartCS)
      *))?
256   | (value=StringLiteral)
257   ) '}'
258   )
```

```
259   | ( (atPre?='@' 'pre')?
260   ({InvocationExpCS.nameExp=current} '(' (
261   argument+=NavigatingArgCS (argument+=NavigatingCommaArgCS)*
262   (argument+=NavigatingSemiArgCS (argument+=NavigatingCommaArgCS)*
263   (argument+=NavigatingBarArgCS (argument+=NavigatingCommaArgCS)*)?
264   )? ')'
265   )?
266   )
267   )
268   )
269   ;
270 NavigatingArgCS returns NavigatingArgCS:
271   name=NavigatingArgExpCS (':' ownedType=TypeExpCS ('=' init=ExpCS)?)?;
272   // Type-less init is an illegal infix expression
273
274 NavigatingBarArgCS returns NavigatingArgCS:
275   prefix='|' name=NavigatingArgExpCS (':' ownedType=TypeExpCS ('=' init=
      ExpCS)?)?;
276 // Type-less init is an illegal infix expression
277
278 NavigatingCommaArgCS returns NavigatingArgCS:
279   prefix=',' name=NavigatingArgExpCS (':' ownedType=TypeExpCS ('=' init=
      ExpCS)?)?;
280   // Type-less init is an illegal infix expression
281
282 NavigatingSemiArgCS returns NavigatingArgCS:
283   prefix=';' name=NavigatingArgExpCS (':' ownedType=TypeExpCS ('=' init=
      ExpCS)?)?;
284   // Type-less init is an illegal infix expression
285
286 NavigatingArgExpCS returns ExpCS: // Intended to be overridden
287   ExpCS
288 ;
289
290 IfExpCS returns IfExpCS:
291   'if' condition=ExpCS
292   'then' thenExpression=ExpCS
293   'else' elseExpression=ExpCS
294   'endif';
295
296 LetExpCS returns LetExpCS:
297   'let' variable+=LetVariableCS (',' variable+=LetVariableCS)*
298   'in' in=ExpCS;
299
300 LetVariableCS returns LetVariableCS:
301   name=UnrestrictedName (':' ownedType=TypeExpCS)? '=' initExpression=
      ExpCS;
302
303 NestedExpCS returns NestedExpCS:
```

```
304   '(' source=ExpCS ')';
305
306 SelfExpCS returns SelfExpCS:
307   {SelfExpCS} 'self';
```

Listing B.6: Extended OCL grammar in Xtext

An example of activity diagram for QML/Cs Language can be seen in Fig **??** and Fig **??**. Measurement designer can declare measurements and import context models for the measurements, application designer on the other hand, can import application model, mapping model and provides an operation, and apply measurement to a specific operation of his/her application. In addition, application designer can place constraints over the applied measurement.

B.2 Complete Example Specifications of a Meta-model for QML/CS

In this appendix we have gathered the full Meta-Depth textual definition for QML/CS meta-model.

B.2.0.1 Defining QML/CS on a Multi-level modelling with MetaDepth

Meta-depth is a framework for deep meta-modelling that developed by de Lara and Guerra [30]. It permits the representation of a meta-model to allow an arbitrary number of ontological and linguistic levels and the dual instantiation. Listing 4.1 shows a meta-model for QML/CS meta-model containing classes of main concepts is defined in MetaDepth. The idea is specifying a three-level meta-modelling architecture where the top-most level contains the definition of class diagrams and potency 2 (QML/CS meta-model in Listing 4.1).

```
1  Model Metamodel@2 {
2
3   abstract Node Type {
4    typeName:String{id};
5    isAbstract@1 :boolean=false;
6   }
7
8   Node Class: Type {
9    name:String;
10   classes:Class[0..*];
11   attribute: Attribute[0..*];
12   operations:Operation[0..*];
```

```
13   associationEnd:AssociationEnd[0..*];
14   associations:Association[0..*];
15   statesMachineModel: StateMachineModel[0..*];
16  }
17
18  Node DataType: Type {
19   name:String;
20  }
21
22  Node Parameter {
23   name : String;
24   type : Class;
25  }
26
27  Node Measurement {
28   name:String;
29   formalParameters:Parameter[*];
30   dateType: DataType;
31   oclExpression: OclExpression[*];
32  }
33  Node Service{
34   name:String;
35   serOperations:Class[0..*];
36  }
37  Node Component{
38   name:String;
39   comOperations:Class[0..*];
40  }
41  Node Attribute {
42   name:String;
43  }
44  Node Operation {
45   name:String;
46  }
47  Node Association {
48   name:String;
49   associationEnd: AssociationEnd[*];
50  }
51  Node AssociationEnd {
52   name:String;
53   iscomposition:boolean = false;
```

```
54   isaggregation:boolean = false;
55  }
56  Node StateMachineModel {
57   name:String;
58   states: States[1..*];
59   transitions: Transition[1..*];
60  }
61  Node State {
62   name:String;
63  }
64  Node Transition {
65   name:String;
66   src: State;
67   tgt: State[*];
68  }
69  Node MappingModel {
70   name:String;
71   applicationModel: ApplicationModel;
72   contextModel: ContextModel;
73   classMapping: ClassMapping[*] ;
74   stateMachineModel:StateMachineModelMapping[*];
75   stateMapping: StateMapping[*] ;
76   transitionMapping: TransitionMapping[*] ;
77  }
78  ClassMapping{
79   name:String;
80   src: Class;
81   tgt: Class;
82   classSMM:StateMachineModelMapping;
83 }
84  StateMachineModelMapping{
85   name:String;
86   src: StateMachineModel;
87   tgt: StateMachineModel;
88 }
89  Node StateMapping {
90   name:String;
91   src: State ;
92   tgt: State;
93 }
94  Node TransitionMapping {
```

```
 name:String;
 transitionsrc: Transition;
 transitiontgt: Transition[*];
}
Node ApplicationModel{
 name:String;
 statesMachineModel: StateMachineModel;
 class: Class[*];
}

Node ContextModel {
 name:String;
 statesMachineModel: StateMachineModel ;
 classes: Class[*] ;
}

Node AssociationEndList {
 name:String;
}

Node AbstractResource {
 name:String;
 demandType: Class;
 resourceModel: ContextModel ;
 cpacitylimit: CapacityLimit ;
 resourceservice: ResourceService ;
}

 Node ResourceService {
 name:String;
 class: Class [*];
}

 Node CapacityLimit {
 name:String;//
 class: Class [*];
}

Node ConcreteResource {
  name:String;
 Class: Class[*];
```

```
136   abstractResource: AbstractResource;
137   cpacitylimit: CapacityLimit;
138  }
139
140  Node Container {
141   name:String;
142   helperVariables: HelperVariables[1..*];
143   parameters: Parameter[0..*];
144   service: Class;
145   component: Component;
146   resource: AbstractResource;
147  }
148
149  Node HelperVariables {
150   name:String;
151   dateType: DataType;
152  }
153
154  Node Component {
155   name:String;
156   classes: Class [*];
157  }
158
159  Node System {
160   name:String;
161   comResContainInstances: ComResContainInstances[*] ;
162   container: Container;
163  }
164
165  Node ComResContainInstances {
166   name:String;
167   component:Component;
168   resource:AbstractResource;
169   container:Container;
170  }
171
172  abstract Node TypedElement{
173   name:String;
174  }
175
176  abstract Node OclExpression:TypedElement{
```

```
177  name:String;
178 }
179
180 abstract Node CallExp:OclExpression{
181  appliedElement: OclExpression[*];
182 }
183
184 Node VariableExp:OclExpression{
185  referringExp:Variable;
186 }
187
188 Node Variable{
189  initializedElement:OclExpression[*];
190  representedParameter:Parameter[0..*];
191 }
192
193 abstract Node FeatureCallExp: CallExp {
194 }
195
196 Node MeasurementCallExp: FeatureCallExp {
197  oclExprs: OclExpression[*];
198  referedMeasurement: Measurement;
199 }
200
201 Node NavigationCallExp: FeatureCallExp {
202  oclExprs: OclExpression[*];
203  referredProperty: AssociationEnd[*];
204 }
205
206  Node PropertyCallExp: FeatureCallExp {
207  oclExprs: OclExpression[*];
208  referredProperty: Attribute[*];
209 }
210
211 Node OperationCallExp: FeatureCallExp {
212  oclExprs: OclExpression[*];
213  referredOperation: Operation[*];
214 }
215
216 Node CapacityLimitCallExp: FeatureCallExp {
217  oclExprs: OclExpression[*];
```

```
218   referedCapacityLimit: CapacityLimit;
219  }
220
221   Node ResourceServiceCallExp: FeatureCallExp {
222   oclExprs: OclExpression[*];
223   referedResourceService: ResourceService;
224  }
225
226 }
```

Listing B.7: A meta-model for QML/CS in Meta-Depth

In this way, in the next meta-level we can build models of QML/CS (e.g., Service, ServiceOperation, ContextModel, Resource and ResponseTime in Listing 4.2), Listing 4.3 and in the bottom meta-level we can build object diagrams or instances of these models which includes OCL expression.

```
1  Metamodel System {
2
3  abstract Node TypedElementM{
4   name:String;
5  }
6
7  abstract Node OclExpressionM:TypedElementM{
8   name:String;
9  }
10
11 abstract Node CallExpM :OclExpressionM{
12  appliedElement: OclExpressionM[0..*];
13 }
14
15 Node ConsExpM:OclExpressionM{
16  referringConExpM:IntegerConsExpM;
17 }
18
19 Node IntegerConsExpM{
20  name:String;
21 }
22
23 abstract Node FeatureCallExpM: CallExpM {
24 }
25
```

```
26 Node NavigationCallExpM: FeatureCallExpM {
27  oclExprs: OclExpressionM[*];
28  referredAssociationEndDemand: CpuDemand;
29  referredAssociationEndInvoassocaions: Invocations;
30 }
31
32 Node PropertyCallExpM: FeatureCallExpM {
33  oclExprs: OclExpressionM[*];
34  referredPropertyTaskId: idTask;
35  referredPropertyTaskDemandWect: wcet;
36  referredPropertyTaskDemanDeadline: deadline;
37  referredPropertyTaskDemanPriod: period;
38 }
39
40 Node OperationM{
41 name : String;
42 }
43
44 Node OperationCallExpM: FeatureCallExpM {
45  oclExprsArgs: OclExpressionM[*];
46  oclExprsSrc: OclExpressionM[*];
47  referredOperation: OperationM[*];
48 }
49
50 Node RTMeasurementCallExp: FeatureCallExpM {
51  rtargs : ServiceOperation[1];
52  referedMeasurement: ResponseTime;
53 }
54
55 Class ServiceOperation {
56  name="ServiceOperation";
57  assoEnd: Invocations[1];
58 }
59
60 ContextModel RtContextModel{
61  name="ResponseTimeContextModel";
62  ServOpStateMM:ServOpStateMachineModel;
63  class: ServiceOperation;
64 }
65
66 StateMachineModel ServOpStateMachineModel{
```

```
67   name="ServiceOperation StateMachineModel";
68   transition1: RequestArrival;
69   transitions2: StartRequest;
70   transitions3: FinishRequest;
71  }
72
73  State Idle {
74   name="Idle";
75  }
76
77  State RequestAvailable {
78   name="Request Available";
79  }
80
81  State HandlingRequest {
82   name="HandlingRequest";
83  }
84
85  Transition RequestArrival {
86   name="RequestArrival";
87   src: Idle;
88   tgt: RequestAvailable;
89  }
90
91  Transition StartRequest {
92   name="StartRequest";
93   src: RequestAvailable;
94   tgt: HandlingRequest;
95  }
96
97  Transition FinishRequest{
98   name="FinishRequest";
99   src: HandlingRequest;
100  tgt: Idle;
101  tg2:RequestAvailable;
102 }
103
104 MappingModel MappingAppToCtx{
105  name="
        MappingStateMachineModelsOfApplicationAnDcontextModels
        ";
```

```
106   applicationModel: ApplicationCounter;
107   contextModel: RtContextModel;
108   stateMapping: StateMapping1;
109   stateMapping: StateMapping2;
110   stateMapping: StateMapping3;
111   transitionMapping: TransitionMapping1;
112   transitionMapping: TransitionMapping2;
113   transitionMapping: TransitionMapping3;
114  }
115
116  StateMapping StateMapping1 {
117   name="StateMapping1";
118   src: OpIdle;
119   tgt: Idle;
120  }
121
122  StateMapping StateMapping2 {
123   name="StateMapping2";
124   src: OpRequestAvailable ;
125   tgt: RequestAvailable;
126  }
127
128  StateMapping StateMapping3 {
129   name="StateMapping3";
130   src: OpHandlingRequest;
131   tgt: HandlingRequest;
132  }
133
134  TransitionMapping TransitionMapping1{
135   name="TransitionMapping1";
136   transitionsrc: OpRequestArrival;
137   transitiontgt: RequestArrival;
138  }
139
140  TransitionMapping TransitionMapping2{
141   name="TransitionMapping2";
142   transitionsrc: OpStartRequest;
143   transitiontgt: StartRequest;
144  }
145
146  TransitionMapping TransitionMapping3{
```

```
147   name="TransitionMapping3";
148   transitionsrc: OpFinishRequest;
149   transitiontgt: FinishRequest;
150  }
151
152  ApplicationModel ApplicationCounter {
153   name="Operation StateMachineModel";
154   OpStateMM:OpStateMachineModel;
155   transition1: OpRequestArrival;
156   transitions2: OpStartRequest;
157   transitions3: OpFinishRequest;
158  }
159
160  StateMachineModel OpStateMachineModel{
161   name="ServiceOperation StateMachineModel";
162   transition1: OpRequestArrival;
163   transitions2: OpStartRequest;
164   transitions3: OpFinishRequest;
165  }
166
167  State OpIdle {
168   name="Operation Idle";
169  }
170
171  State OpRequestAvailable {
172   name="Request Available";
173  }
174
175  State OpHandlingRequest {
176   name="Operation HandlingRequest";
177  }
178
179  Transition OpRequestArrival {
180   name="Operation RequestArrival";
181   src: OpIdle;
182   tgt: OpRequestAvailable;
183  }
184
185  Transition OpStartRequest {
186   name="StartRequest";
187   src: OpRequestAvailable;
```

```
188   tgt: OpHandlingRequest;
189  }
190
191  Transition OpFinishRequest {
192   name="FinishRequest";
193   src: OpHandlingRequest;
194   tgt1: OpIdle;
195   tg2:OpRequestAvailable;
196  }
197
198  Service SystemService {
199   name="SystemService";
200   serOp: ServiceOperation [0..*];
201  }
202
203  Parameter op{
204   name = "ResponseTimeop";
205   type1 : ServiceOperation[1]{type};
206  }
207
208  AssociationEnd Invocations{
209   name="Invocations";
210  }
211
212  Class OperationCall{
213   attrStart:start;
214   attrEnd:end;
215   assoEnd: Invocations[1];
216  }
217
218  Attribute start{
219   name="Start";
220  }
221
222  Attribute end{
223   name="End";
224  }
225
226  Operation last{
227  name="last";
228  }
```

```
229
230 Measurement ResponseTime{
231  name="response_time";
232  opParam: op[1]{formalParameters};
233  opParam2: op1[1]{formalParameters};
234  dataType: Real;
235  spec: VariableExp0;
236 }
237
238 VariableExp VariableExp0{
239  referredVariable:spec;
240 }
241
242 Variable spec{
243  initializedElement:operationCallExpr1;
244 }
245
246 OperationCallExp operationCallExpr1{
247  src:operationCallExpr3;
248  args:operationCallExpr2;
249  referredOp: minus;
250 }
251
252 Operation minus{
253  name="-";
254 }
255
256 OperationCallExp operationCallExpr2{
257  src:NavigationCallExp2;
258  arg:propertyCallExpr1;
259  referredOp:last;
260 }
261
262 OperationCallExp operationCallExpr3{
263  src:NavigationCallExp1;
264  arg:propertyCallExpr2;
265  referredOp:last;
266 }
267
268 PropertyCallExp propertyCallExpr1{
269  referredAttr:start;
```

```
}

PropertyCallExp propertyCallExpr2{
 referredAttr:end;
}

NavigationCallExp NavigationCallExp1{
 src:VariableExp1;
 assoc:Invocations;
}

NavigationCallExp NavigationCallExp2{
 src:VariableExp2;
 assoc:Invocations;
}

VariableExp VariableExp1{
 referredVariable:variable1;
}

VariableExp VariableExp2{
referredVariable:variable1;
}

Variable variable1{
 representedParameter:op;
}

DataType Real{
 name="real";
}

Attribute idTask{
 name="id";
}

Attribute wcet{
 name="wcet";
}

Attribute deadline{
```

```
311  name="deadline";
312 }
313
314 Attribute period{
315  name="period";
316 }
317
318 AbstractResource CPU{
319  name="CPU";
320  type:CpuTask;
321  cpuModel:CpuModel;
322  cpuServuce:CpuResourceService;
323 }
324
325 ContextModel CpuModel{
326  name="CPUModel";
327  functionTimeAlloted: TimeAlloted;
328  functionCanHandle: CanHandle;
329 }
330
331 StateMachineModel CpuStateMachineModel{
332  name="CPUStateMachineModel";
333  function: TimeAlloted;
334 }
335
336 Class TimeAlloted{
337  name="timeAlloted";
338  scheduledTasks:scheduledTask;
339 }
340
341 Class CanHandle{
342  name="canHandle";
343  scheduledTasks:scheduledTask;
344  oclexpr:operationCallexp11;
345 }
346
347 OperationCallExp operationCallexp11{
348  src:tupleLiteralPart1;
349  arg:ResponseTimeContainer;
350  referredtoOp:equal;
351 }
```

```
352
353  OperationCallExp operationCallexp12{
354   src:tupleLiteralPart2;
355   arg:ResponseTimeContainer;
356   referredtoOp:equal;
357  }
358
359  OperationCallExp operationCallexp13{
360   src:tupleLiteralPart3;
361   arg:ExecutionTimeContainer;
362   referredtoOp:equal;
363  }
364
365  TupleLiteralExp tupleLiteralExp1{
366   referredToTuple:tupleLiteralPart1;
367   referredToTuple:tupleLiteralPart2;
368   referredToTuple:tupleLiteralPart3;
369  }
370
371  TupleLiteralPart tupleLiteralPart1{
372   referredtoAt1:period;
373  }
374
375 TupleLiteralPart tupleLiteralPart2{
376   referredtoAt2:deadline;
377  }
378
379  TupleLiteralPart tupleLiteralPart3{
380   referredtoAt2:wcet;
381  }
382
383  CapacityLimit CpuCapacityLimit{
384   tasks: CpuTask[*];
385   oclexpr: operationCallExpr17[*];
386  }
387
388  ResourceService CpuResourceService{
389   tasks: CpuTask[*];
390   always1:VariableExp3;
391   always2:VariableExp4;
392  }
```

```
393
394 VariableExp VariableExp3{
395  referredVariable:always1;
396 }
397
398 VariableExp VariableExp4{
399  referredVariable:always2;
400 }
401
402 Variable always1{
403  initializedElement:operationCallExpr4;
404 }
405
406 OperationCallExp operationCallExpr4{
407  src:capacityLimitExp1;
408  arg:resourceServiceCallExp1;
409  referredOp:GreaterThanOrEqualTo;
410 }
411
412 Operation GreaterThanOrEqualTo{
413  name="greater than or equal to";
414 }
415
416 CapacityLimitCallExp capacityLimitExp1{
417  arg:navigationCallExp4;
418  referredCapacityLimit:CpuCapacityLimit;
419 }
420
421 ResourceServiceCallExp resourceServiceCallExp1{
422  arg:navigationCallExp3;
423  referredResService:CpuResourceService;
424 }
425
426 NavigationCallExp navigationCallExp3{
427  referredNav:CpuDemand;
428 }
429
430 NavigationCallExp navigationCallExp4{
431  referredNav:CpuDemand;
432 }
433
```

```
434 Variable always2{
435  initializedElement:collect;
436 }
437
438 IterateExp collect{
439  body:includesAll;
440  src:navigationCallExp5;
441  iterator:t;
442 }
443
444 VariableExp variableExp5{
445  referredVraible: t;
446 }
447
448 Variable t{
449 }
450
451 NavigationCallExp navigationCallExp5{
452  src:variableExp5;
453  referredNav:scheduledTask;
454 }
455
456 IterateExp includesAll{
457  src:navigationCallExp6;
458  body:operationCallExp5;
459  iterator:t;
460 }
461
462 NavigationCallExp navigationCallExp6{
463  src:variableExp6;
464  referredNav:CpuDemand;
465 }
466
467 VariableExp variableExp6{
468   referredVariable: t;
469 }
470
471 OperationCallExp operationCallExp5{
472  src:navigationCallExp7;
473  arg:operationCallExp6;
474  referredOp: and;
```

```
475 }
476
477 NavigationCallExp navigationCallExp7{
478  referredNav:CpuDemand;
479 }
480
481 Operation and{
482  name="And";
483 }
484
485 Operation size{
486  name="size";
487 }
488
489 OperationCallExp operationCallExp6{
490   src:navigationCallExp8;
491   arg:operationCallExp7;
492   referredOp:size;
493 }
494
495 NavigationCallExp navigationCallExp8{
496  referredNav:scheduledTask;
497 }
498
499 OperationCallExp operationCallExp7{
500  src:navigationCallExp9;
501  referredOp:equal;
502 }
503
504 Operation equal{
505  name="=";
506 }
507
508 NavigationCallExp navigationCallExp9{
509  src:operationCallExp8;
510  referredNav:CpuDemand;
511 }
512
513  OperationCallExp operationCallExp8{
514  src:forAll;
515  referredOp:and;
```

```
516 }
517
518 IterateExp forAll{
519  src:navigationCallExp6;
520  body:operationCallExp9;
521  iterator:t;
522 }
523
524 OperationCallExp operationCallExp9{
525  src:operationCallExp10;
526  arg:propertyCallExp3;
527  referredOp:GreaterThanOrEqualTo;
528 }
529
530 OperationCallExp operationCallExp10{
531  arg:navigationCallExp11;
532  referredOp: TimeAlloted;
533 }
534
535 PropertyCallExp propertyCallExp3{
536  src:navigationCallExp10;
537  referredPro:wcet;
538 }
539
540 NavigationCallExp navigationCallExp10{
541  src:variableExp7;
542 }
543
544 VariableExp variableExp7{
545  referredVar:t;
546 }
547
548 VariableExp variableExp8{
549  referredVar:t;
550 }
551
552 NavigationCallExp navigationCallExp11{
553   src:variableExp8;
554 }
555
556 ConcreteResource RmsCpu{
```

```
557  name="RmsCpu";
558  specialize:CPU;// specialize
559  cpuCapacitylimit: CpuCapacityLimit;
560 }
561
562 Class CpuTask{
563  scheduledTasks:scheduledTask;
564 }
565
566 AssociationEnd scheduledTask{
567  attributeId:idTask;
568  demands: CpuDemand;
569 }
570
571 AssociationEnd CpuDemand{
572  name="CPUDemand";
573  attributeWcet:wcet;
574  attributeDeadline:deadline;
575  atttributePeriod:period;
576 }
577
578 AssociationEnd ContainerCpuDemand{
579  name="ContainerCpuDemand";
580  wcet:ResponseTimeContainer;
581  deadline:ResponseTimeContainer;
582  period:ExecutionTimeContainer;
583 }
584
585 Container SimpleContainer{
586  name="simpleContainer";
587  containerParameters: ResponseTimeContainer;
588  helperVariables: ExecutionTimeContainer;
589  service: ContainerService;
590  component: C;
591  resource: CPU;
592 }
593
594 Parameter ResponseTimeContainer{
595  name="ResponseTime";
596  dateType: Real;
597 }
```

```
598
599  HelperVariables ExecutionTimeContainer {
600   name="ExecutionTimeHelperVariable";
601   dateType: Real;
602 }
603
604  Measurement ExecutionTime{
605   name="Execution_time";
606   opParam: op1[1]{formalParameters};
607   dataType: Real;
608  }
609
610  Component C{
611   name="CounterComp";
612   provides: op1;
613   always:variableExp9;
614  }
615
616  VariableExp variableExp9{
617   referredVar:always3;
618  }
619
620  Variable always3{
621   initializedElement:operationCallExpr10;
622  }
623
624  Operation lessThan{
625   name="<";
626  }
627
628  MeasurementCallExp measurementCallExp2{
629   referredMeas:ExecutionTime;
630  }
631
632  OperationCallExp operationCallExpr10{
633   src:measurementCallExp2;
634   arg:helperVariableExp1;
635   referredOp:lessThan;
636  }
637
638  HelperVariableExp helperVariableExp1{
```

```
639  referredtoHelpvar:ExecutionTimeContainer;
640 }
641
642 Parameter op1{
643  name = "ExecutionTimeop1";
644  type1 : ComponentOperation[1]{type};
645 }
646
647 Class ComponentOperation{
648  name="ComponentOperation";
649 }
650
651 Class ContainerService {
652  name="Service";
653  oclexpr: operationCallexp14;
654 }
655
656 OperationCallExp operationCallexp14{
657  src:operationCallExpr16;
658  arg:VariableExp5;
659  referredtoOp:GreaterThanOrEqualTo;
660 }
661
662 VariableExp VariableExp5{
663  referredVariable:always4;
664 }
665
666 Variable always4{
667  initializedElement:operationCallExpr15;
668 }
669
670 MeasurementCallExp measurementCallExp1{
671  referredMeas:ResponseTime;
672 }
673
674 OperationCallExp operationCallExpr15{
675  src:measurementCallExp1;
676  arg:ResponseTimeContainer;
677  referredToOp:lessThan;
678 }
679
```

```
680 OperationCallExp operationCallExpr16{
681  src:ExecutionTimeContainer;
682  arg:ResponseTimeContainer;
683  referredToOp:lessThan;
684 }
685
686 OperationCallExp operationCallExpr17{
687  src:iterate;
688  arg:operationCallExpr18;
689  referredToOp:lessThanOrEqualTo;
690 }
691
692 Operation lessThanOrEqualTo{
693  name="Less than or equal to";
694 }
695
696 OperationCallExp operationCallExpr18{
697  src:operationCallExpr21;
698  arg:operationCallExpr19;
699  referredToOp:multiplication;
700 }
701
702 Operation multiplication{
703  name="*";
704 }
705
706 OperationCallExp operationCallExpr19{
707  src:propertyCallExp4;
708  arg:consExp1;
709  referredToOp:minus;
710 }
711
712 ConsExp consExp1{
713 referringConExp:integerConsExp1;
714 }
715
716 IntegerConsExp integerConsExp1{
717  name="1";
718 }
719
720 Attribute sqrt{
```

```
721  name="sqrt";
722 }
723
724 PropertyCallExp propertyCallExp4{
725  src:integerConsExp2;
726  arg:operationCallExpr20;
727  referredPro:sqrt;
728 }
729
730 ConsExp consExp2{
731 referringConExp:integerConsExp2;
732 }
733
734 IntegerConsExp integerConsExp2{
735  name="2";
736 }
737
738 OperationCallExp operationCallExpr20{
739  src:navigationCallExp12;
740  referredToOp:size;
741 }
742
743 NavigationCallExp navigationCallExp12{
744  referredNav:CpuDemand;
745 }
746
747 NavigationCallExp navigationCallExp13{
748  referredNav:CpuDemand;
749 }
750
751 IterateExp iterate{
752  body:operationCallExpr22;
753  src:navigationCallExp14;
754  iterator:t;
755 }
756
757 NavigationCallExp navigationCallExp14{
758  src:variableExp12;
759  referredNav:CpuDemand;
760 }
761
```

```
762 Operation division{
763  name="/";
764 }
765
766 OperationCallExp operationCallExpr21{
767  src:navigationCallExp13;
768  referredToOp:size;
769 }
770
771 OperationCallExp operationCallExpr22{
772  src:operationCallExpr23;
773  arg:propertyCallExp5;
774  referredOp:division;
775 }
776
777 PropertyCallExp propertyCallExp5{
778  src:variableExp10;
779  referredOp:deadline;
780 }
781
782 Operation addition{
783  name="+";
784 }
785
786 OperationCallExp operationCallExpr23{
787  src:operationCallExpr23;
788  arg:propertyCallExp6;
789  referredOp:addition;
790 }
791
792 PropertyCallExp propertyCallExp6{
793  src:variableExp11;
794  referredOp:wcet;
795 }
796
797 VariableExp variableExp10{
798  referredVar:t;
799 }
800
801 VariableExp variableExp11{
802  referredVar:t;
```

```
803  }
804
805  VariableExp variableExp12{
806   referredVar:t;
807  }
808
809 }
```

Listing B.8: Model Definition for QML/CS in Meta-Depth

Listing 3 shows an instance of QMLCS Model, namely a class diagram named QMLCSModel which declares fourteen Nodes and one association.

```
 1 System CounterApplication{
 2 SystemService counter{
 3   name="counter";
 4   serOp=getData;
 5 }
 6 ServiceOperation getData {
 7   name = "getData";
 8 }
 9 op getDataopParameter {
10   type1 = getData;
11 }
12 ResponseTime getDataResponseTime{
13  opParam = getDataopParameter;
14 }
15 OperationM lessThan{
16  name="<";
17 }
18 OperationCallExpM operationCallExpM1{
19  oclExprsSrc=measurementCallExp1;
20  oclExprsArgs=consExpM1;
21  referredOperation=lessThan;
22 }
23 RTMeasurementCallExp measurementCallExp1{
24   RTargs=getData;
25   referedMeasurement=getDataResponseTime;
26 }
27 ConsExpM consExpM1{
28  referringConExpM=integerConsExpM1;
29 }
```

```
30 IntegerConsExpM integerConsExpM1{
31  name="60";
32 }
33 }
```

Listing B.9: Instances *responseTime* context Model Definition for QML/CS in Meta-Depth